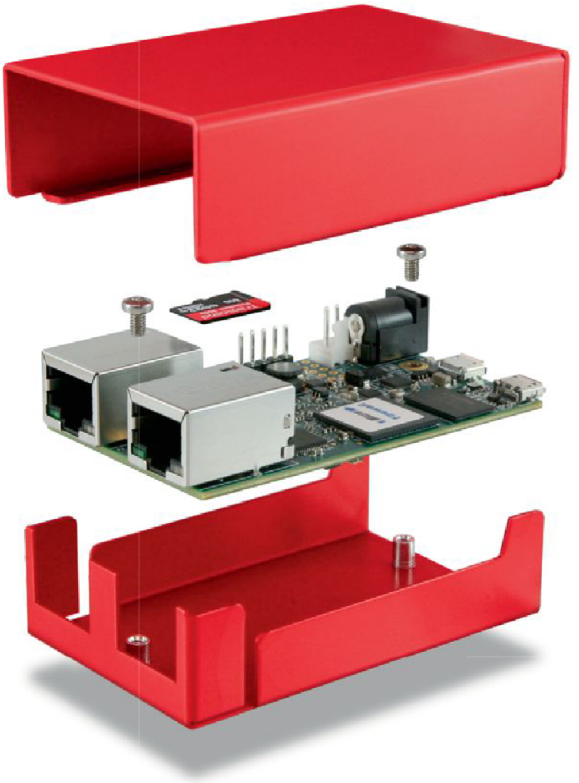
FreeBSD™ JOURNAL
Sept/Oct 2016

Welcome FreeBSD 11

GELI, ZFS, TCP & TLS IMPROVEMENTS!



INTRODUCING THE SG-1000

microFirewall

$149

Includes pfSense® Gold, a $99 value.

pfSense®

You asked. We delivered! The new Netgate® SG-1000 microFirewall is a cost-effective, state-of-the-art, ARM®-based, pfSense® Security Gateway appliance. The SG-1000 comes with dual 1Gbps Ethernet ports, enabling maximum throughput exceeding 100 Mbps. The ARM Cortex®-A8 in the TI AM3552 SoC and DDR3L RAM combine to facilitate low power consumption while maintaining performance. The SG-1000 comes in a lightweight and durable anodized aluminum case. It's credit-card sized form-factor allows it to be easily tucked away, but you'll be proud to show it off.

- Pre-loaded with pfSense software, ready to use right out of the box.
- Easy GUI management. Manage pfSense software settings through our web based GUI.
- Comes with a one year subscription to pfSense Gold ($99 value).
- No artificial limits or add ons required to make your system fully functional.
- Flexible configuration and support for VPN, load balancing, reporting and monitoring.

Netgate®

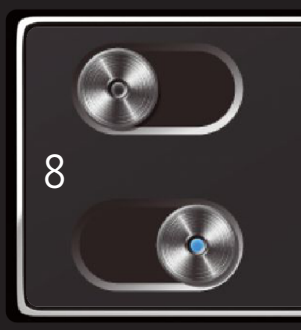
Shop now at the Netgate store or authorized partners worldwide.

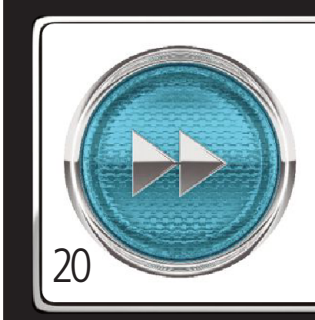
www.netgate.com/products/sg-1000.html

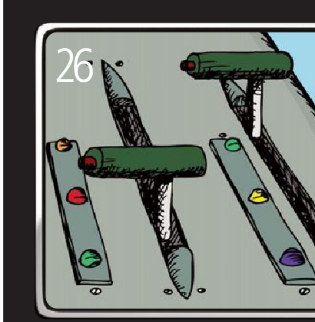
FreeBSD™ JOURNAL

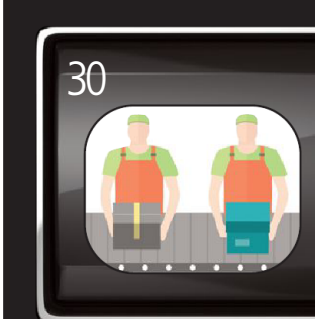
Sept/Oct 2016

Table of Contents









S&W Publishing LLC
PO BOX 408, BELFAST, MAINE 04915

Publisher • Walter Andrzejewski
walter@freebsdjournal.com

Editor-at-Large • James Maurer
jmaurer@freebsdjournal.com

Art Director • Dianne M. Kischitz
dianne@freebsdjournal.com

Office Administrator • Michael Davis
davism@freebsdjournal.com

Advertising Sales • Walter Andrzejewski
walter@freebsdjournal.com
Call 888/290-9469

FreeBSD Journal (ISBN: 978-0-615-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).

Published by the FreeBSD Foundation,
5757 Central Ave., Suite 201, Boulder, CO 80301
ph: 720/207-5142 • fax: 720/222-2350
email: info@freebsdfoundation.org

LETTER from the Board

"This one goes to 11."

There are certain generational jokes we cannot avoid. For those of us who grew up with, or at least have seen, the film *This is Spinal Tap*, the number 11 always elicits the line that is the title of this month's letter.

The 11th major release of FreeBSD is now out and John Baldwin covers what is new and exciting in 11 in his article in this issue.

Making 11 major releases of a complete operating system over a span of 23 years is a major achievement. Thousands of fingers have touched the docs, ports, and source trees over the years, and even more have consumed, repurposed, and deployed the code onto desktops, servers, laptops, and embedded systems. Getting to where we are today is nothing short of amazing to those of us who continue to work on FreeBSD. This is why I want to take a slight departure from my typical letter to talk about a particular set of folks who are working to help the FreeBSD Project and need your help.

Most readers know that the *FreeBSD Journal* is supported not only by your subscriptions, but largely by the FreeBSD Foundation, of which I am one of the directors. I joined the board of the FreeBSD Foundation because I knew that I could apply my non-technical skills to help the FreeBSD community in ways that engineers might not normally do, including producing this magazine, the production of teaching and marketing materials, and fundraising. It is on the fundraising front that I wish to talk to you here.

The FreeBSD Foundation exists to support FreeBSD through the funding of many different parts of the FreeBSD ecosystem. We give direct financial support to BSD-related conferences, including AsiaBSDCon, BSDCan, EuroBSDCon, and many newer conferences that you might not have heard of as yet. In addition, the Foundation sponsors the growing number of developer summits held throughout the year. Although many have predicted that the Internet and associated technologies would reduce the need for face-to-face meetings, that turns out to have been false. In fact, much of the best work being done on FreeBSD begins at these developer meetings.

The Foundation employs engineering resources—from the head of release engineering, to the full-time, and contract staff who fill the technical holes in our software firmament not otherwise being filled in by the community. This support allows for the timely, stable, and reliable releases you have come to expect. The Foundation

has also hired a full-time marketing director to help the Project reach those who have yet to learn how great FreeBSD is, and to help the community grow.

Finally, the Foundation funds much of the hardware on which the Project now runs. Whether that is release builds, package servers, or the growing continuous integration infrastructure, the Foundation has been here to provide the hundreds of thousands of dollars required over the last several years to make sure that the software produced by the community can easily and efficiently be consumed by the entire world.

Unlike in the Linux world, the FreeBSD Project is completely open. Anyone is welcome to join and there is no tax levied by the FreeBSD Foundation to attend events such as developer meetings or vendor summits. Unfortunately, that openness has a downside. As the FreeBSD Project has grown, so have its needs, and that means that the Foundation is always in fundraising mode. Although I could appeal to each and every one of you to send a donation to the Foundation, that is not what the final paragraph of this letter is about.

Help Support FreeBSD https://www.freebsdfoundation.org/donate

I know many of this *Journal*'s readers work in technology companies. Quite a few, and I'm assuming the majority of you, work at companies that use FreeBSD every day in some capacity. This is where we need you. Help us by figuring out how to connect the FreeBSD Foundation to the companies you work with, or know of, that have an interest in the continuing success of FreeBSD. Whether it is through me directly (gnn@freebsdfoundation.org), or with the Foundation's Executive Director (deb@freebsdfoundation.org), those connections are absolutely necessary to the continuing success of both the FreeBSD Foundation and the FreeBSD Project.

I know that what I'm asking may not come naturally to those who work in the trenches of engineering. "Talk to management?! That's the last thing..." and so we have a few suggestions on this web page (https://www.freebsdfoundation.org/donate/donation-letter/) of how you might start that conversation.

Thank you
George Neville-Neil
Editor in Chief of the *FreeBSD Journal*
Member of the Board of Directors of the FreeBSD Foundation

SEE TEXT ONLY

By John Baldwin

WELCOME
To FreeBSD 11

The FreeBSD system is constantly changing. FreeBSD 11 brings new features and fixes from two and a half years of active development. Some of these changes have been merged to recent 10.x releases such as 10.2 and 10.3, but most of them are brand new in 11.

Desktop and Laptop

FreeBSD 11 offers a variety of improvements for desktop and laptop users. First, a new system console driver is enabled by default. This driver is less VGA- and x86-centric than previous drivers. Rather than depending on BIOS ROM support for VGA text modes, the console renders text in software on framebuffers. This supports VGA adapters via graphics modes as well as the UEFI framebuffer. It also supports graphics adapters that disable VGA compatibility when using

higher resolution graphics modes such as modern Intel GPUs. Software text rendering allows the console driver to render any glyph, which in turn enables UTF-8 support. In addition, the in-kernel graphics drivers include native support for Intel graphics adapters on systems with fourth-generation Core ("Haswell") processors.

FreeBSD 11 includes broader support for wireless networks. The new iwm(4) driver supports Intel integrated wireless adapters using the 3160, 3165, 7260, 7265, and 8260 chipsets via 802.11a/b/g. These adapters are used on most laptops with a fourth generation or later Intel Core processor. The iwn(4) driver (used on laptops with earlier Core processors) now supports 5-Ghz channels as well as 802.11n. The ath(4) driver for Atheros adapters supports newer, 802.11n-capable adapters with full 802.11n support in both station and AP modes. The bwn(4) driver for Broadcom BCM43xx wireless adapters now supports devices with an N-PHY (BCM4312 and BCM4321 chipsets). These adapters support 802.11n on 5-Ghz channels. The rsu(4) and urtwn(4) drivers for Realtek USB adapters now fully support 802.11n on 2.4-Ghz channels.

Support for UEFI has been improved in 11 as well. The FreeBSD amd64 (also known as x86_64 or x64) install media will now boot from either UEFI or legacy mode. UEFI systems can now boot directly from a ZFS filesystem. Other booting improvements for both UEFI and legacy systems including support for ZFS boot environments and whole-disk encryption via GELI are covered in more detail in Allan Jude's article in this issue.

The bhyve hypervisor has several new features in FreeBSD 11. Guest machines can now be started with UEFI firmware rather than using a user-space boot loader. This permits guest operating systems that support UEFI to use a native boot process. In addition, bhyve supports a graphics framebuffer when using UEFI along with additional device emulations for keyboard and mouse input. These emulated devices are backed by a VNC server. This allows guest operating systems to use a graphical interface. Users interact with these guests via a VNC client. Together with other fixes, these changes permit Microsoft Windows to run as a guest in a bhyve virtual machine. In addition, bhyve in FreeBSD 11 includes a device emulation for the Intel 82545 network adapter. This permits the use of networking with operating systems that do not support VirtIO devices. In particular, Windows can be used out-of-the-box without requiring additional VirtIO device drivers during installation.

Finally, FreeBSD 11 includes support for PCI-express native HotPlug. This includes handling of runtime insertion and removal of ExpressCard adapters in laptops as well as runtime insertion and removal of PCI-express adapters in HotPlug-capable slots.

Enterprise

FreeBSD isn't purely a desktop OS, of course, and 11 includes several new features for the enterprise. Along with support for PCI-express HotPlug, 11 also includes support for PCI Single-Root I/O Virtualization (SR-IOV). This includes the ability to create virtual functions (VFs) on supported device drivers. These VFs can be passed to virtual machines executing under the bhyve hypervisor to enable direct access to hardware for I/O requests.

The iSCSI stack introduced in FreeBSD 10 has gained several improvements in 11. FreeBSD now supports iSCSI Extensions for RDMA (iSER) providing more efficient zero-copy I/O to SCSI data buffers. The cxgbei(4) driver supports hardware-accelerated offload of iSCSI connections on TOE-capable Chelsio adapters. Finally, the reroot utility provides a means for booting a system with an iSCSI root filesystem.

The local storage stack has also seen a raft of changes in 11. FreeBSD now includes a zfsd daemon to handle automatic activation of hot spare devices and other ZFS-related events not handled in the kernel. The sesutil(8) utility supports management of disk enclosures, and the mpsutil(8) utility permits management of LSI Fusion-MPT 2 (mps(4)) and Fusion-MPT 3 (mpr(4)) SAS/SATA controllers. FreeBSD 11 includes support for pluggable disk I/O scheduling in the CAM layer. A CAM front-end for NVMe disks is present in 11 that can be used instead of the nvd(4) driver to enable CAM-specific behavior with NVMe disks.

FreeBSD 11 also includes new drivers for various hardware. The OFED Infiniband stack has been updated to version 2.1 including support for RoCE. The ixl(4) driver supports Intel XL710 40Gb Ethernet adapters, and the mlx5en(4) driver supports Mellanox ConnectX-4 and ConnectX-4LX adapters.

ARM

FreeBSD 10.1 was the first release to ship release images for supported FreeBSD/arm systems. FreeBSD 11 has expanded support for ARM-based systems in several ways.

First, ARM kernels now include a global shared-page exported to all user processes. As a result, user processes on ARM systems now use a non-executable stack. In addition, user processes are able to fetch the current time of day without system call overhead.

Second, the default floating point ABI for 32-bit ARMv6+ has been changed from soft-float to hard-float. This enables increased floating-point performance on modern processors.

Third, the virtual memory system in FreeBSD's kernel supports transparent, 1-megabyte super pages on 32-bit ARMv6+ processors. As with support for super pages on x86, this reduces the overhead of TLB misses. ARM systems are even able to map the text segment of the C runtime library with a super page. (The text segment on x86 is too small to use a super page mapping.)

Fourth, FreeBSD 11 includes support for more systems. Support for several Allwinner SoCs has been added including support for the Banana Pi, Cubieboard 1, and Cubieboard 2. 11 also includes install images for the PandaBoard and Raspberry Pi 2 systems.

Finally, FreeBSD 11 adds support for 64-bit ARMv8 processors via the FreeBSD/arm64 platform. 11 boots out of the box on Cavium ThunderX systems and support for additional systems will be available in future releases. The arm64 platform includes a global shared-page enabling use of non-executable stacks and fast time-of-day queries as on the 32-bit ARM platform. Support for super pages is already present in HEAD and will be available in FreeBSD 11.1. The FreeBSD package system includes over 20,000 prebuilt packages for FreeBSD/arm64 that are updated on a regular basis.

RISC-V

RISC-V is a new, open-source instruction set architecture. Initially motivated by computer architecture research, it is freely available for all types of use including commercially produced CPUs. FreeBSD 11 includes a new FreeBSD/riscv platform that supports the 64-bit RISC-V architecture. The RISC-V ISA for kernel mode is still in flux, but FreeBSD 11 supports version 1.9 of the draft privileged ISA specification. FreeBSD/riscv boots to multiuser in both the Spike simulator and QEMU emulator.

Developer Friendly

FreeBSD 11 is more developer friendly than ever. The llvm toolchain in the base system (including clang and lldb) has been updated to release 3.8.0. lldb is now included as part of the base system on FreeBSD/amd64 and FreeBSD/arm64. The C++ runtime library (libc++) has also been updated, which includes support for C++14.

The entire base system is now built with debug symbols. These can be installed either at install time or after install. These symbols permit developers to inspect state and single-step through base system libraries as well as application code.

Threading support has also received several improvements. The POSIX threads library now supports process-shared primitives such as mutexes and condition variables. An implementation of robust mutexes has also been added to the thread library. In addition, internal improvements to the debugging subsystem permit more robust debugging of multithreaded processes.

Finally, the DTrace tracing utility is now supported on more platforms in 11, including ARM, MIPS, and RISC-V.

Conclusion

FreeBSD's community has put a ton of effort into FreeBSD 11 over the past two and a half years, and it shows. Thank you to everyone who has contributed, whether by testing snapshots, reporting bugs, submitting patches, maintaining patches, working with users on social media, organizing conferences, etc. We hope you enjoy FreeBSD 11, and we look forward to your contributions to FreeBSD 12! ●

JOHN BALDWIN joined the FreeBSD Project as a committer in 1999. He has worked in several areas of the system including SMP infrastructure, the network stack, virtual memory, and device driver support. John has served on the Core and Release Engineering teams and organizes an annual FreeBSD developer summit each spring.

Simplify your Data Center

Meet iXRack — A powerfully flexible rack-scale architecture that takes the guesswork out of building large scale data center applications.

- **Converged Infrastructure**
- **Customizable for Virtualization, Big Data, Cloud & Hyperscale**
- **Up to 70% Lower TCO Than AWS**
- **Scalable and Repeatable Deployments**

For more information on iXRack, visit **iXsystems.com/iXRack** today

IMPROVING High-Bandwidth TLS in the FreeBSD Kernel

BY
RANDALL
STEWART &
SCOTT LONG

In our 2015 paper, "Optimizing TLS for High-Bandwidth Applications in FreeBSD," we demonstrated the cost of TLS encryption on high–bandwidth video serving on Netflix's OpenConnect Appliance (OCA) [1] and explored methods for reducing that cost via in-kernel encryption. The results showed that such a concept is feasible, but only a small performance gain was achieved. In this article we describe the improvements made since then to reduce the cost of encryption. We also compare the performance of several different bulk encryption implementations.

INTRODUCTION

The importance of Transport Layer Security (TLS) [2] continues to grow as businesses and their customers come to appreciate the value of communication privacy. Netflix announced in 2015 that it would start on the path to encrypting the video and audio playback sessions of its streaming service in an attempt to help users protect their viewing privacy [3]. Enabling this comes with a significant computational cost to the OpenConnect serving platform, so work continues in exploring and implementing new ways to optimize TLS bulk encryption and thus lower capital and operational costs.

An OCA is a FreeBSD-based appliance that serves movies and television programming to Netflix subscribers. Confidential customer-data-like payment information, account authentication, and search queries are exchanged via an encrypted TLS session between the client and the various application servers that make up the Netflix infrastructure. The audio and video objects are statically encrypted by Digital Rights Management (DRM) that is pre-encoded into the objects prior to them being distributed to the OCA network for serving.

The Netflix OpenConnect Appliance is a server-class computer based on an Intel 64-bit Xeon CPU and running FreeBSD and Nginx 1.9. The platform evolves yearly as commodity components increase in capability and decrease in price; the most recent generation of the platform holds between 10TB and 200TB of multimedia objects, and can accommodate anywhere from 10,000 to 40,000 simultaneous long-lived TCP sessions with customer client systems. The servers are also designed to deliver between 10Gbps and 40Gbps of continuous bandwidth utilization.

Communication with the client is over the HTTP protocol, making the system essentially into a large static-content web server.

Until 2015, these audio and video objects were not encrypted on a per-session basis. Netflix is now in the process of revising and releasing its client playback software to include support for TLS playback encryption, with the goal of updating all playback devices that are both capable of being upgraded and have the computational capacity to support TLS decryption. By the end of 2016 we expect the majority of streaming sessions to be using TLS encryption.

As the number of client sessions doing TLS grows across the OpenConnect network, demand increases on the server side to accommodate these sessions. Building on the results from our work in 2015 [4], we looked for ways to reduce the cost of encryption on our deployed fleet of hardware as well as reduce the number of servers needed to accommodate future growth. This investigation covered three areas: what is the ideal cipher for bulk encryption, what is the best implementation of the chosen cipher, and are there ways to improve the data path to and from that cipher implementation?

BULK CIPHER SELECTION

The Cipher Block Chaining (CBC) is commonly used to implement bulk data encryption for TLS sessions as it is well studied and relatively easy to implement. However, it comes with a high computational cost as the plaintext data must be processed twice, once to generate the encrypted output, and once to generate the SHA hash that verifies the integrity of the encrypted data. The AES-GCM cipher, based on Galois/Counter Mode (GCM) [5], provides adequate data protection and does not require that the plaintext be processed twice. It is included in TLS 1.2 and later, and is available in all modern versions of OpenSSL and its derivatives. Decryption is also computationally cheap, and this combined with ubiquitous availability makes it attractive to both client and server platforms.

We decided that we would transition our TLS platform to prefer GCM, and fall back to CBC only for primitive clients that couldn't be upgraded to support GCM. We estimate that once TLS is rolled out to our entire network only a small percentage of playback sessions will use CBC.

CIPHER IMPLEMENTATION

Synthetic performance testing of the OpenCrypto Framework AES-CBC cipher showed it to be less performant than the equivalent implementation from OpenSSL. We also needed to investigate AES-GCM performance, but found that OpenSSL 1.0.1 as of early 2015 did not have an AESNI-optimized implementation for it. Our search for alternatives led us to BoringSSL [6], which had a well-performing AESNI implementation of AES-GCM.

In mid–2015 we were introduced to the Intel Intelligent Storage Acceleration Library (ISA-L) [7]. It provided an implementation of AES-GCM that was hand-tuned for specific Intel model families and their instruction sets. Testing showed it to be an improvement over the BoringSSL ciphers. Results are included below.

One drawback to the ISA-L was that it was written in the YASM dialect, which is not directly compatible with the GCC and LLVM toolchain assemblers in FreeBSD. That required us to modify the kernel makefile infrastructure in a rudimentary way to allow YASM to be called as an external assembler on the needed source files, as so:

```
compile-with "/usr/local/bin/yasm -g dwarf2
-f elf64 ${INTELISAINCLUDES} -o ${.TARGET}
$S/contrib/intel_isa/aes/${.PREFIX}.asm"
```

DATA PATH IMPROVEMENTS

Our initial work in 2015 [4] used the AESNI implementation built into FreeBSD and the Open Crypto Framework (OCF) to perform bulk encryption. The results in our previous paper showed a small improvement in performance, but not nearly the results we had hoped to gain. We knew of several areas where our results could be improved, including:

- Extra copies were being made during kernel data processing due to the encrypt in-place requirement of our AESNI implementation.
- The nginx calls into the TLS code were not passing in correct flags with the `sendfile(2)` call. This meant that hot content was not being properly cached.
- Many times during processing an mbuf chain was walked to gain addresses for encryption; this constant walking of the mbuf linked lists caused added overhead and further polluted the CPU cache.

We decided to pass in to our new ciphers an

array of pointers to encrypt from and to, i.e., an iovec. This iovec array would be filled in during the initial setup of the sendfile call, as each page was set up for I/O, thus eliminating the need for traversing a linked list of mbufs. We also redesigned the mbuf allocation routines to have the ability, as allocation occurred, to include this new "mbuf map."

Since a large part of our data was being encrypted, we also designed a new special mbuf zone that required less overhead during allocation. A typical one-page mbuf required three separate allocations (one for the mbuf, one for the refcount, and one for the page). We redesigned this to make the page and the mbuf an indivisible unit where FreeBSD's UMA would allocate a page and mbuf together during the UMA's initialization routine and the UMA constructor would only be used to reset pointers within the tied entity. We also embedded the reference count within the mbuf. This required some small tricks with copies (we don't actually free the original mbuf until all copies are free), but proved quite effective at reducing mbuf overhead.

Switching to the iovec array forced us to abandon the OpenCrypto Framework API and access the cipher routines directly. We still wanted to be able to fall back to OpenCrypto for testing purposes, so we created a layer that abstracts the memory allocation and iovec handling for low-level cipher access while still allowing interoperability with OpenCrypto. The translation is transparent to the upper layers and is selectable at runtime. This work also gave us the chance to find and fix codepaths that were making unnecessary copies of data. We also fixed the incorrect sendfile flag usage.

RESULTS

After adding all the improvements, we deployed our new firmware on three different machines. These machines were fed live traffic while gathering CPU and bandwidth measurements during busy hours. The same software was used in all measurements, the only difference being changes to the configuration so that the software would:

- Disable all sendfile enhancements and use just OpenSSL, reading from the file and writing the encrypted data down the TCP connection.
- Use the sendfile enhancement with the encryption set to use boringSSL.
- Use the sendfile enhancement with the encryption set to use Intel's ISA library.

Thus each machine provided us with three sets of results. The machine types were as follows:

- Rev H storage (HDD) platform, CPU E5-2650Lv2 at 1.7Ghz with 20 cores (Hyperthreaded) the cpu class being an Ivy Bridge Xeon.
- Rev F Flash (SSD) cache platform, CPU E5-2697v2 at 2.7Ghz with 24 cores (Hyperthreaded) the cpu class being an Ivy Bridge Xeon.
- Rev N Flash (SSD) cache platform, CPU E5-2697v3 at 2.6Ghz with 28 cores (Hyperthreaded) the cpu class being a Haswell Xeon.

Each set of results will be labeled Rev H, Rev F, or Rev N with the test undergone. We show approximately one hour of traffic during a busy period. For visual clarity, the legends have been removed from the graphs; the green x plots are bandwidth in Gbps, and the red + plots are CPU system usage percentage.

We see in Figure 1 what happens when only OpenSSL is used. The CPU limits we have set are the standard 80%; however, the storage cache is disk bound hitting between 60% and 65% CPU and topping our performance out at about 12–12.5Gps of serving traffic. The sendfile feature adds considerable improvement as we see in the next two figures.

The results of BoringSSL, in kernel using sendfile, are seen in Figure 2. The CPUs tend to be used a bit more (55%–70% CPU utilization)

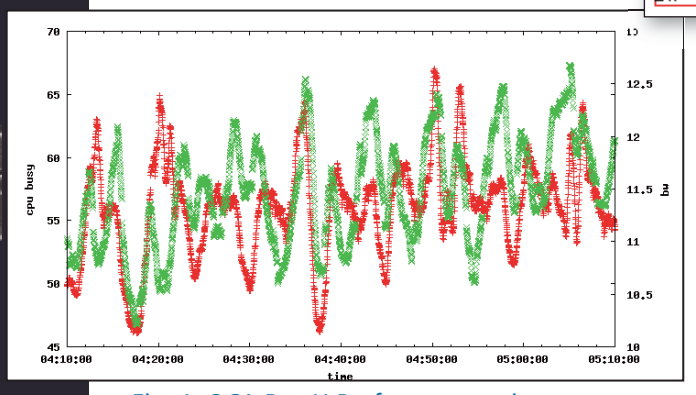


Fig. 1. OCA Rev H Performance using user space OpenSSL

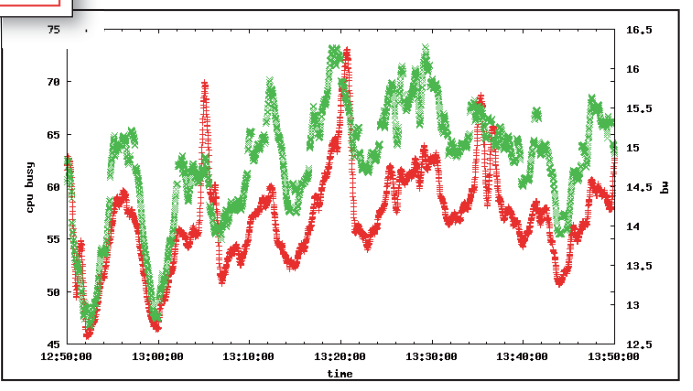


Fig. 2. OCA Rev H Performance using in kernel BoringSSL

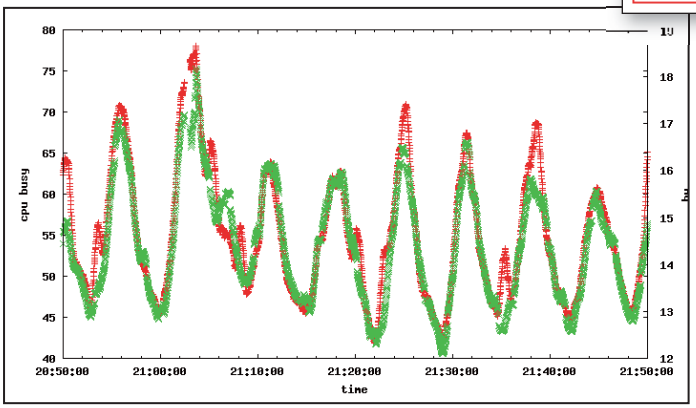


Fig. 3. OCA Rev H Performance using in kernel ISA\ in kernel ISA

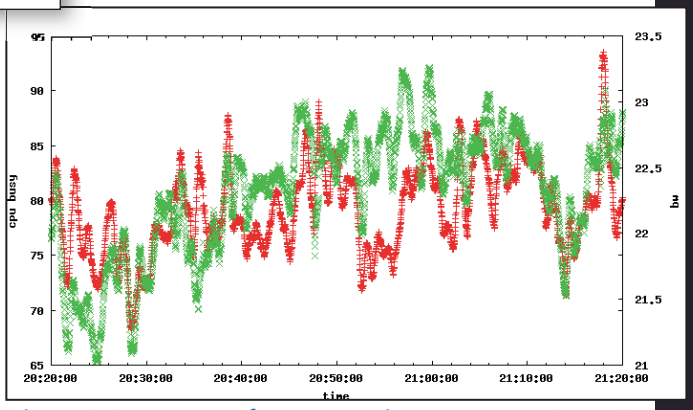


Fig. 4. OCA Rev F Performance using user space OpenSSL

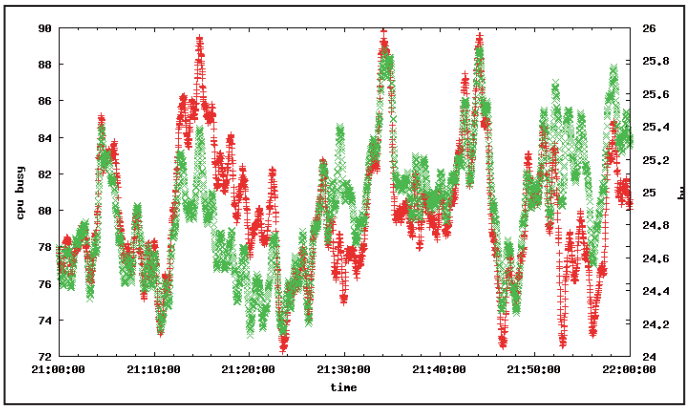


Fig. 5. OCA Rev F Performance using in kernel BoringSSL

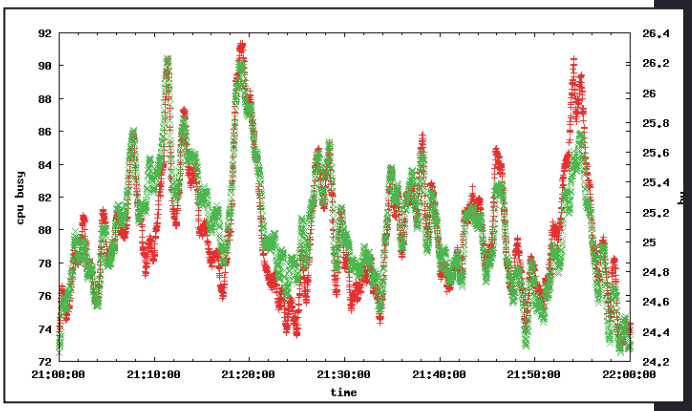


Fig. 6. OCA Rev F Performance using in kernel ISA

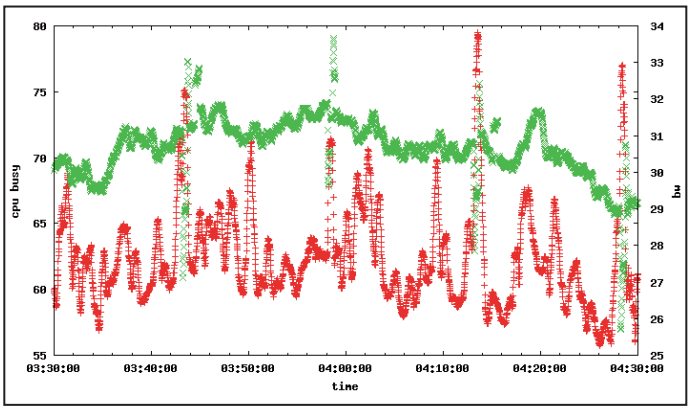


Fig. 7. OCA Rev N Performance using user space OpenSSL

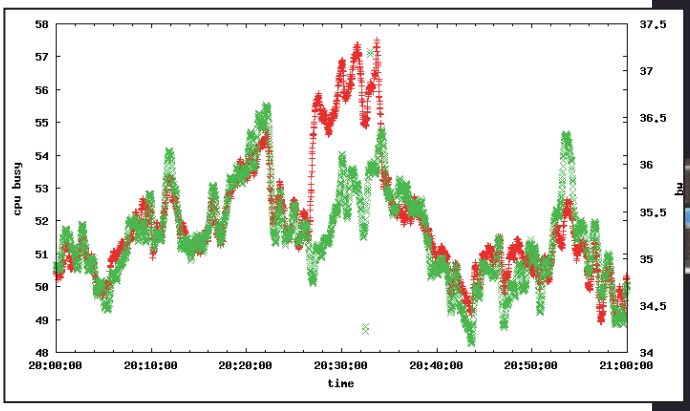


Fig. 8. OCA Rev N Performance using in kernel BoringSSL

with overall output performance increasing to 15–16Gps. This is what we expect with use of the sendfile call to help streamline the I/O.

For final comparison, we put in place the ISA library, again in kernel with sendfile. Results can be seen in Figure 3 and show another improvement moving us to as much as 18G but generally holding around 16–16.5G.

In Figure 4 we see OpenSSL, this time hitting maximum CPU. This is because SSDs have a significantly greater I/O capacity and so we no longer hit the disk limits seen in the Rev H. We see that running with an average of 80% CPU we maintain somewhere between 22 to 23Gbps. This gives us our baseline to compare any improvements.

In Figure 5 we see the results of using the kernel encryption with sendfile and BoringSSL. Here we are able to maintain between 25 and 25.5Gbps while maintaining our goal of 80% CPU utilization.

The ISA library shown in Figure 6 gives us a slight improvement over the previous results getting us again around 25–25.5G, but tending to stay toward the higher end of the range.

Both the Rev F and Rev H are Ivy Bridge machines (v2 CPUs). We anticipate better performance out of a Haswell machine (v3 CPU). Our Rev N shows promising results in the next set of figures.

Interestingly, the OpenSSL results seen in Figure 7

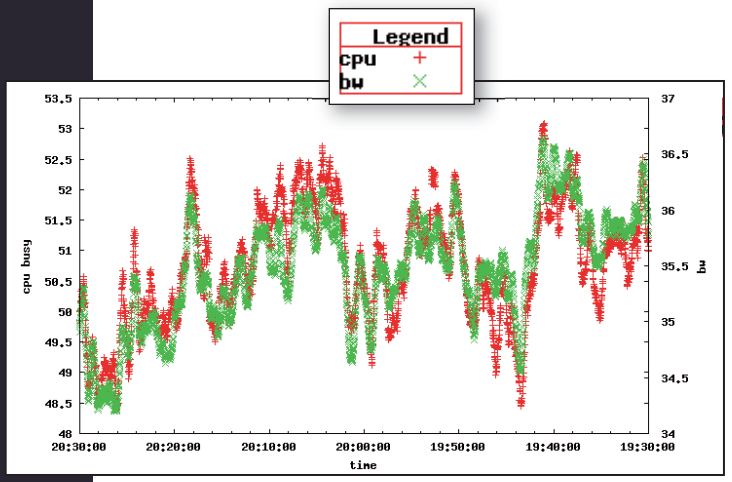


Fig. 9. OCA Rev N Performance using in kernel ISA

do not reach the full CPU target of 80%. Checking the machine health statistics, we found that the SSDs had reached their maximum at around 29–30Gps that we see maintained in the graph.

In Figure 8 we see us reach the interface's maximum capacity of 35.5–36Gbps, with the CPU tending to stay around 53% with a burst up to 57% at one point.

The ISA library results can be seen in Figure 9 and show similar results to what we see in the BoringSSL case with the exception that our CPU use is tending to stay toward 50.5%. The results are tabulated in Table I.

	RevF		RevH		RevN	
	cpu%	BW	cpu%	BW	cpu%	BW
Baseline	60–65	12–12.5	80	22–23	70–75	29–30
BoringSSL	55–70	15–16	80	25–25.5	53	35.5–36
ISA-L	55–70	15–16.5	80	25.5	50.5	35.5–36

Table I. CIPHER COMPARISON CHART

SUMMARY AND FUTURE DIRECTIONS

With our latest work, overall performance has improved as much as 30%, a vast improvement from our original results. Still left unexplored is use of offload cards to assist in our efforts to encrypt all data. One question we have is whether a card can be used in our hardware design in such a way that it takes less CPU and PCIe bandwidth than just running the AESNI instructions themselves. As new generations of Intel CPUs become available, it is possible that the cost of feeding data to an auxiliary card and collecting the encrypted results will be more than the actual AESNI instructions themselves.

We were also forced to set aside the OpenCrypto Framework in order to achieve certain optimization. Similar optimizations might be useful for other crypto consumers in the kernel, so we will explore ways to extend the OCF API as we work to move the code upstream to FreeBSD.

Encrypting in software still requires that data pages get touched by the CPU, causing CPU cache pollution. The data is almost never reused, so the caching is wasted and needlessly evicts other system data that could be reused. We are currently investigating fine-grained use of cache-control features of the Intel CPUs to limit the amount of data that is put into the last–layer caches during encryption. We are also working with the Intel ISA-L team to develop routines that use uncached load/store assembly opcodes for data movement through the cipher.

ACKNOWLEDGMENTS

The authors would like to thank the Intel ISA-L team for reaching out to us to assist in achieving our performance goals, and David Wolfskill and Warner Losh for their support and reviews. We would also like to thank the entire Netflix OpenConnect team for providing an outstanding environment for this work.

ADDENDUM

Shortly after this paper was originally published at AsiaBSDCon in March 2016, Intel publically released the E5-v4 (Broadwell Xeon) series of CPUs. We were already evaluating prerelease samples of these CPUs under our TLS workload, but could not include our results in the paper at that time due to the pre-release nature of the hardware. We also started experimenting with the Mellanox ConnectX-4 100Gb Ethernet adapter as a replacement for our 40Gb adapters, and HGST SN100 NVMe drives as a replacement for our SATA SSD storage.

The 2697-v4 CPUs bumped up the CPU core count to 16, an increase of 2 cores from the v3 predecessor. More importantly, the v4 family added a feature called Cache Allocation Technology (CAT) that allows the OS to exercise fine-grained control over the retention of data pages in the L2 and L3 caches. This was attractive since most of the data moving through the CPU

under our workload is TLS data that is never directly inspected by the CPU once the encryption phase is over and usually never gets reused. Marking this data as non-cachable meant that the limited cache resources could be used for OS data and control structures instead of throw-away TLS data. Supporting CAT policy required only straight-forward code modifications to the TLS socket layer in the kernel. The results are shown in Table II.

	Without SSL		With SSL		With SSL + CAT	
	Gbps	cpu%	Gbps	cpu%	Gbps	cpu%
E5-2697-v3	89	80	55	80	–	–
E5-2697-v4	93	80	58	80	62	80

Table II. NETFLIX VIDEO STREAMING

These results build on the data from Table I, with the notable difference being that thanks to moving to a 100Gb network interface, the CPU was now the limiting factor on performance, not network port bandwidth. An important data point not shown in the table is that the v4 CPU was also tested in the baseline configuration with userland OpenSSL and no kernel data path. In this configuration, only 26Gbps was achieved before maxing out the CPU. Thus, the difference between the baseline and optimized configuration represents a 136% increase in performance on the v4 CPU and a 106% improvement compared to the v3 CPU at baseline. We attribute the lower gain relative to the v3 to the extra CPU cores creating additional resource and cache contention.

Our results show that it is feasible to design and deploy a system running FreeBSD that can fill 2x40Gbps or 1x100Gbps network attachments to a reasonable capacity with 100% TLS traffic using modest hardware components. Performance research continued in 2016, and our results from that will be shared in a future paper. ●

REFERENCES

[1] Netflix, "Netflix Open Connect," https://openconnect.netflix.com/en/. (2016)

[2] T. Dierks, E. Rescorla. "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246. (August 2008)

[3] M. Watson. "HTTPS performance experiments for large scale content distribution," https://lists.w3.org/Archives/Public/www-tag/2015Apr/0027.html. (April 2015)

[4] R. Stewart, J. M. Gurney, S. Long. "Optimizing TLS for High–Bandwidth Applications in FreeBSD," https://people.freebsd.org/ rrs/asiabsd 2015 tls.pdf. (March 2015)

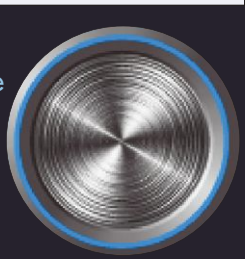
[5] D. A. McGrew, J. Viega. "The Galois/Counter Mode of Operation (GCM)," http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/ proposedmodes/gcm/gcm-spec.pdf. (2005)

[6] Google, "BoringSSL," https://boringssl.googlesource.com/boringssl/. (2016)

[7] Intel, "Optimizing Storage Solutions Using the Intel Intelligent Storage Accelerations Library," https://software.intel.com/en-us/articles/optimizing-storage-solutions-using-the-intel-intelligent-storage-acceleration-library/. (2015)

RANDALL STEWART currently works for Netflix Inc. as a Senior Software Engineer. His current duties include optimizing and enhancing FreeBSD's network stack within the Netflix Open Connect Appliance. Previously Stewart was a Distinguished Engineer at Cisco Systems. In other lives he has also worked for Adara, Motorola, NYNEX S&T, Nortel and AT&T Communication. Throughout his career he has focused on operating system development, fault tolerance, and call control signaling protocols. Stewart is also a FreeBSD committer having responsibility for the SCTP reference implementation within FreeBSD as well as working with FreeBSD's tcp stack to enhance and extend it.

SCOTT LONG has been a Senior Software Engineer at Netflix Inc. since 2011, focusing on FreeBSD performance and system architecture. Previously he was at Yahoo, Adaptec, and a handful of tech startups. Long has been a FreeBSD committer since 2000 and a user since 1993. Outside of work, he enjoys aviation, hiking, and skiing with his family in Colorado.



SEE TEXT ONLY

GELI and ZFS IMPROVEMENTS

A lot of new things are coming in FreeBSD 11, but this article will focus on a number of contributions I personally had a hand in. The focus of these changes is improving the way FreeBSD systems are booted, and how system updates are managed.

BCache Improvements

For many years the FreeBSD loader has implemented a simple block cache to improve performance by avoiding physical reads of the same sector multiple times. When this cache was originally developed, most systems booted from a single disk, or a RAID volume that was exposed to the operation system as a single logical disk. With the advent of ZFS, it is very common for the loader to need to access multiple disks in order to read all of the data required to load the kernel and boot the operating system. An IllumOS developer, Toomas Soome, who is working on porting the FreeBSD loader to IllumOS to replace the ancient version of grub that they currently use, developed a number of improvements to the block cache and contributed those back to FreeBSD. The improvements include growing the cache from 16 KB to 16 MB, implementing a rudimentary read-ahead cache, and making it support multiple devices. The 16 MB cache is divided evenly amongst the number of devices detected at boot. This can likely be optimized more in the future.

The read-ahead feature takes advantage of the fact that for spinning media, reading a modest number of contiguous sectors takes almost no additional time compared to reading just a single

sector. When the next sector is read, it is returned from the cache. If two different areas of the disk are being read concurrently, this can provide an immense performance gain. Additionally, support was added for caching data fetched from CD and DVD devices, which previously were not cached. This change in particular made a large difference for people who boot FreeBSD installer images via remote media systems such as IPMI. The kernel-loading portion of booting from a virtual CD (an ISO image) via IPMI over a LAN was reduced from 27 seconds to 7 seconds, thanks to the read-ahead cache. The difference becomes even more stark when the remote case is considered. Without the block cache, when installing FreeBSD on a remote server with 60ms latency, a single 2K block is read from the CD at a time, and the next block is not requested until that read operation for the previous block completes, 60+ms later. With the new read-ahead cache and other bcache improvements, the time to load the kernel from a remote CD dropped from over 12 minutes to under 5 minutes. Now when I have to build out a new server in Singapore with 225ms of latency, I am very grateful for Toomas's hard work. After a number of other contributions, Toomas Soome has officially joined the FreeBSD Project as a committer.

ZFS Boot Environment Menu

The basics of what boot environments are and how they work have been covered before (see the September/October 2015 issue of this publication). To quickly recap, a "Boot Environment" (BE) is an alternative root filesystem. ZFS provides low-cost snapshots and clones, meaning that before a change to the operating system like an upgrade, the operator can take a snapshot to be able to revert the system to in the case of a failed upgrade. These snapshots can be used to create clones, which are writable snapshots. Having a selection of these clones to choose from allows the operator to try different versions of the operating system, without disturbing the currently installed version. Some filesystems like /home are common between the different boot environments. This ensures that even as the operator tries different versions of FreeBSD, their same home directory is available in all of them. This also means updated contents of the home directory are not lost if an upgrade is reverted.

Here is a look at the boot environments that exist on my development machine:

NAME	USED	AVAIL	REFER	MOUNTPOINT
zroot/ROOT	11.3G	2.51T	392M	legacy
zroot/ROOT/9.0_router	392M	2.51T	392M	/
zroot/ROOT/9.1_before_upgrade	16.0K	2.51T	1.92G	/
zroot/ROOT/9.1_freebsd	822M	2.51T	822M	/
zroot/ROOT/9.1_pcbsd	16.0K	2.51T	2.05G	/
zroot/ROOT/9.2_beta1	848M	2.51T	848M	/
zroot/ROOT/9.2_pcbsd	16.0K	2.51T	2.10G	/
zroot/ROOT/before_10_stable_2014-09-11	16.0K	2.51T	2.68G	/
zroot/ROOT/10_stable_2014-03-24	647K	2.51T	3.35G	/
zroot/ROOT/11_1100093	16.0K	2.51T	3.35G	/
zroot/ROOT/11_r295359_zfsdebug	368K	2.51T	3.38G	/
zroot/ROOT/default	8.95G	2.51T	3.42G	/

As you can see, I have upgraded this machine in-place from 9.0-RELEASE through to 11-CURRENT. There is even a BE from when I was debugging a ZFS memory consumption issue in 11-CURRENT. In this BE the kernel contains a lot of additional DTrace probes to try to track the source of the problem. At the end of a debugging session, I could reboot back to my standard system, which was 10-STABLE. Being able to flip back and forth with just a reboot, while maintaining a common /home directory, and not requiring fragmenting my free disk space across separate partitions, is very powerful.

Originally, the only way to manage BEs was with the beadm(1) utility. This posed an obvious problem—if the system failed to boot, how do you run the beadm utility to change the active boot environment to the previous working system? PC-BSD adapted a solution similar to that used in IllumOS, where the beadm utility generates a list of BEs and stores them in a configuration file that is read by the GRUB boot loader. While this mostly works, it requires using a non-standard boot loader, and can run into problems when the config file gets out of sync with reality. The GRUB config file loads its modules from the original boot environment that was active when

GRUB was set up, which can cause the system to be unbootable if that BE is removed and the configuration file is not properly updated.

An easier and more reliable solution was needed. The FreeBSD loader already has the ability to read from ZFS and to list the available datasets, as part of the 'lszfs' loader command. I used this command as a template, and created a new function that populates a set of environment variables with a paginated list of boot environments. The main advantage to this approach is that it reads from the live filesystem, so it is always accurate. With help from Devin Teske, the loader's menu system, written in forth, now displays a menu and allows you to select which boot environment to use as your root filesystem. This was further improved with help from Toomas Soome. One drawback was that the loader menu system was not supported if the system was booted with EFI, because our EFI loader lacked the required serial emulation code. Again, Toomas Soome to the rescue. The missing features were implemented and the ZFS boot environment menu is available in the FreeBSD loader for both BIOS and EFI boot.

GELIBoot

Now ZFS BEs were supported across pretty much any configuration, except when the user opts for full disk encryption. When the pool containing the root filesystem is encrypted, a separate /boot filesystem (either a second ZFS pool, or a small UFS partition) is required. This plaintext partition is where the boot loader, kernel, and modules like GELI and ZFS are stored. The system needs to be able to bootstrap itself, which requires reading the boot loader, which in turn reads and executes the kernel, which loads the module to be able to access the encrypted filesystem. The issue is that this 'two pools' setup, breaks ZFS Boot Environments. When the kernel lives outside of the root filesystem, which is what is snapshotted and cloned to make the various boot environments, it becomes impossible to switch between BEs and load the matching kernel. Even if similar snapshots were managed between the two pools, it would be complex and error prone. A better solution was needed.

Being the naive junior developer that I am, I asked "how hard can it be?" For an initial implementation, a copy of `gptzfsboot` was made with the name `gptgeliboot`. The original idea was to create a single bootcode that could boot from an encrypted UFS or ZFS filesystem. Subtle differences in bits of the `boot2` code, and no clear way to define the behaviour of systems that use both UFS and ZFS filesystems, caused this approach to be abandoned. It was decided to implement optional GELI support in each of the existing GPT bootcodes instead. Initially, it was necessary to determine if the boot partition was GELI encrypted. As with most all GEOM classes, GELI stores its metadata in the very last sector of the provider, which is usually a partition, to avoid conflicting with the backup copy of the GPT partition table that is stored in the last sector of the disk. The task seemed simple—read the partition table, identify the starting LBA of the partition and its size, and read the last sector of that partition. The hardest part about working in the bootcode is that there are no error reporting facilities. There isn't even a `panic()`. Pretty much all there is to work with is `printf()`, and when things go bad, the system just hangs, unless you manage to crash the BTX loader, which will give you a dump of the assembly instruction pointers and the like. This made development very iterative and almost brute force. Make a change, build, install it, reboot, fail, add printf, build, install, reboot, fail, repeat. Of course, you must moderate the quantity of printfs, because there is no pager; once data scrolls off the top of the screen, it is gone forever.

After the partition has been determined to contain encrypted data, the GELI metadata needs to be read to determine which algorithm to use to decrypt it. Then the encrypted copy of the master key must be decrypted with the user-provided password, and only then is it possible to read from the partition. The existing bootcode happened to be structured in such a way that it was relatively easy to add decryption to the regular read functions when GELI was determined to be present on a partition. The first large roadblock was encountered after adding all of the dependencies for GELI (SHA256, SHA512, HMAC, AES -CBC, and AES-CTS). The size of the boot2 file (gptzfsboot) has grown from 47kb to 90kb. The boot1 code (gptldr), a 512 byte block of assembly that loads boot2 into a specific position in memory and executes it, only loaded the first 64kb of boot2. Because this code runs in 16-bit real mode, this is the largest block of data that can be managed at once. After many failed

attempts, and asking for assistance from many senior members of the community, Colin Percival finally solved the problem and extended `gptldr` to load a specified number of 32kb segments of boot2, allowing for it to easily be extended in the future.

In order to reuse existing code from GELI and OpenCrypto (the kernel's AES implementations, used in IPSEC), some changes needed to be made. A few declarations and ifdef's needed to be moved in the GELI code to allow it to be compiled for userspace. Once this was accomplished I set to the task of breaking the OpenCrypto framework up into separate files per algorithm, rather than a single monolithic file. This made it possible to reuse this code rather than importing another copy of AES-XTS for use in the loader.

Much to my surprise, the GELI source code was very elegant, and easy to follow, even by a junior developer who had little prior experience with cryptography. With this work complete, it is now possible to boot FreeBSD from an encrypted UFS or ZFS filesystem, with the only plaintext being the contents of the miniscule freebsd-boot partition (gptboot or gptzfsboot).

At this point, the system can be booted, but it prompts the user for the encryption passphrase an inordinate number of times. The test system was a ZFS mirror of two disks. `gptzfsboot` would prompt for the password for each of the two disks, then load the loader. The loader would then prompt the user for the passphrase for each disk, and load the kernel. At the mountroot prompt, the kernel would prompt for the passphrase for each disk, and finally the system would boot. With more than a few disks, this quickly becomes exceedingly cumbersome. Colin Percival, Devin Teske, and Kris Moore had already been suffering from similar problems and developed a solution. Colin implemented the `kern.geom.eli.boot_passcache sysctl`, which caches the password entered by the user at the mountroot prompt and attempts to reuse it on each new disk that is tested during the boot process. This was extended with Colin's help by Kris Moore, to allow the passphrase entered in the GRUB2 boot loader to be passed via the kernel's environment to GELI, so that if the password was correct, it would avoid re-prompting for the password at the mountroot phase. This avoided the issue where the mountroot password prompt would become buried by late device attach notices. Devin Teske added an option to `loader.conf, geom_eli_passphrase_prompt` that would cause the FreeBSD loader to prompt the user for the GELI passphrase ahead of time, and pass it to the kernel via the environment, the same way PC-BSD's GRUB2 was doing it, again with the goal of avoiding the mountroot prompt. Extra care is taken by the GELI kernel module to zero the passphrase from the environment before single-user mode starts. The password prompt that was implemented in GELIBoot was given a similar caching mechanism whereby it automatically tries the previously entered passphrase, and only if that fails, gives the user three attempts to enter the correct passphrase.

The GELIBoot code needs the passphrase the earliest, in order to read the loader from the encrypted disk. This raised the obvious question, how to pass the passphrase from the `boot2` stage, to the loader. The answer lay in `gptzfsboot`, where a flag is set, `KARGS_FLAGS_EXTARG`, which tells the loader to look for an additional argument after the end of the regular set of arguments. Here, it will find `struct zfs_boot_args`, containing information such as the pool that is being booted, and the root filesystem. The first member of this struct is 'size', which is set to `sizeof(struct zfs_boot_args)`. This allows the loader to safely access newer members of the struct, by first checking that the `offsetof()` the member is not greater than the `sizeof()` the loader's definition of the struct. This allows mismatched versions of the bootcode and loader to continue to work together. When a new member is added to the end of the struct, access to it is guarded by this mechanism. Using this design pattern, a new member was added to the `zfs_boot_args` struct to pass the GELI passphrase from `boot2` to the loader. This too is carefully zeroed as soon as the next boot phase starts.

Work then started on updating the FreeBSD installer to create this configuration for users. There are a number of limitations. The GELIBoot system currently only works with GPT formatted disks. MBR formatted disks are not supported because of a similar restriction in the size of the boot2 code. Originally it was not clear that the ZFS on-disk format allowed a larger bootcode to be installed. After discussion with Toomas Soome, I learned that the ZFS disk label leaves 3.5 megabytes of space for boot code, so it will be

possible to expand the MBR boot2 code. This will likely be required as ZFS grows additional features anyway, so GELIBoot will most likely be extended to include MBR support in FreeBSD 11.1. Currently the FreeBSD EFI boot loader does not support GELI. Another developer, Eric McCorkle, who wrote the original EFI ZFS support, is working on this, and it is also expected to be included in FreeBSD 11.1. However, if you use GPT and boot via the legacy/BIOS method, the installer will create just a single fully encrypted ZFS pool and boot from that. In all other cases, the previous method of using a second plaintext pool to store the loader and kernel is used.

The remaining limitation is also that only GELI passphrases are supported. The GELI master key is encrypted using a key derived by PKCS#5 v2 from the passphrase. GELI also supports using a key file (or the combination of both a passphrase and a key file); however, GELIBoot does not support key files. In the future, support for key files stored on external media such as a USB device is planned. With the UEFI version of GELIBoot, Eric plans to support storing key material in the TPM.

You can read more about my adventures in implementing GELIBoot in the proceedings of AsiaBSDCon 2016, or download my paper here: http://www.allanjude.com/bsd/AsiaBSDCon2016_geliboot_pdf1a.pdf.

Automatic Boot Recovery

FreeBSD contains a toolkit called nanobsd that has been used by many projects, including FreeNAS and pfSense, to build appliances. It contains a firmware-style upgrade feature, where the system is set up with two partitions, and when an upgrade is performed, the new system is installed into the inactive partition, and a partition table flag is set. On the next boot, the partition containing the newer system image will be booted, and the failed flag will be set on it. If it boots successfully, the flags will be removed from the partition. If booting fails, when the system is power cycled, the boot loader will see the failed flag and automatically revert to booting the original image.

There is a desire to have a similar system for ZFS, without requiring separate partitions. The basic idea is to set a pool property, altbootfs, to indicate a new boot environment to attempt to boot. When the boot loader detects the presence of this property, it will mark the partition as failed, and boot from the alternate boot environment. If the system comes up properly, a startup script will remove the failed flag and promote the boot environment to being the default. If the system does not boot correctly, when the next boot is attempted, it will boot the default boot environment, which contains the system from before the upgrade was applied. This will protect appliances and servers from requiring hands-on assistance to recover from a failed upgrade.

ZFS RAID 10

The FreeBSD installer has supported creating mirrored ZFS pools since 10.1; however, if your system contained a large number of disks and you selected the mirror option, the installer would create a single mirror containing all of the disks. While this provides extremely good redundancy with 8 or more drives, this is likely not the user's intent. In FreeBSD 11 the installer includes a RAID-10 option, which creates a ZFS mirror consisting of pairs of disks. An even number of disks must be selected for this option to be available. A system with 8 disks will now create a pool of 4 mirror sets each containing 2 disks. This is the highest performance configuration, especially in terms of IOPS. Mirror sets also provide the most flexibility. Additional disks can be added in pairs, rather than requiring an entire additional RAID-Z vdev. In budget-constrained configurations, this allows small numbers of disks to be purchased when available space runs low. Mirror sets also allow additional space to be gained by replacing smaller disks with larger ones. Once both members of a mirror have been individually replaced and resilvered, the additional space is available. With an 8-disk RAID-Z, all 8 disks would need to be replaced before additional space became available.

And So Much More

The rest of the FreeBSD community has not been idle during this time either; a lot of great work has been done recently. FreeBSD 11 will also include:

- The ZFS ARC is now resizable at runtime, allowing greater control over the amount of memory used by ZFS
- Improved interaction between ZFS and the VFS layer improves responsiveness under memory pressure and allows ZFS to return more memory to the system when needed

- ZFSd is now available; this daemon manages hot spares and automatically reattaching disks that are temporarily detached
- ZFS now supports the SHA512 and Skein checksum algorithms
- The in-kernel SHA2 implementation has been replaced with a much more performant one
- bhyve now supports Windows Guests, and has a VNC backend to provide video console access
- libxo allows a number of utilities in the base system to output JSON, XML, or HTML instead of only plaintext
- Ifconfig now supports multiple output formats, including printing subnet masks in dotted-quad and CIDR notation, and traditionalhex output
- More features are enabled by default, including Netmap and IPSec

More to Come

While 11 is a major milestone, there is plenty more to come. The 11.x branch features a new support model, where each point release is supported for three months after the next point release. This new model will allow more frequent releases and greatly lessen the burden on the Security and Ports teams having to support 2-year-old releases. Many new features will be available in 11.1, although for some of the major changes you'll have to wait for 12. Over the next few months I hope to see a number of new features land:

- Packaged base system
- GELI in the UEFI loader
- Support for more ZFS features in the loader
- ZFS compressed ARC
- ZFS compressed send/recv
- ZFS scrub/resilver speed improvements
- Installer create larger EFI partitions
- Easier management of EFI partition and its contents
- Installer put swap partition before data partition to allow data partition to be easily grown
- Installer optionally install a basic set of packages (graphical environment)
- libucl (standardized configuration file format) config files for many utilities and daemons in base
- libxo in more utilities
- librification of more utilities (ifconfig, netstat)

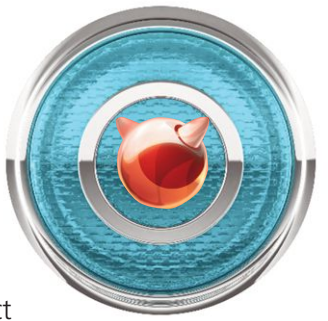
ALLAN JUDE is VP of Operations at ScaleEngine Inc., a global HTTP and Video Streaming CDN, where he makes extensive use of ZFS on FreeBSD. He is also the host of the video podcasts BSDNow.tv (with Kris Moore) and TechSNAP.tv. He is a FreeBSD src and doc committer, and was elected to the FreeBSD Core team in the summer of 2016. Allan is the coauthor of *FreeBSD Mastery: ZFS* and *FreeBSD Mastery: Advanced ZFS* with Michael W. Lucas.

SEE TEXT ONLY

TCP IMPROVEMENTS in FreeBSD 11

BY HIREN PANCHASARA

One of the major obstacles preventing us from experimenting with TCP was fear of the unknown. Because TCP is complex and the code itself was non-modular in its past form, it was difficult to segregate a fix that wouldn't impact the rest of the code. However, it was relatively easy to introduce subtle regressions.



THE TCP ALTERNATIVE STACK FRAMEWORK attempts to fix this problem by modularizing the stack so that anyone can come up with a new TCP stack and have both (or more) stacks coexist on the same machine at the same time and serving different connections. The code is modularized so different TCP operations, like input or output processing, segment handling, timer processing, and socket options parsing, are subdivided into different function blocks and are made accessible with function pointers. Thus, a developer can design an alternate stack with a new way of handling input or output processing and still use the rest of the functionality from the default stack. One can select a stack per connection using socket options which provide the ability to do real A/B testing when experimenting with various TCP features.

The following sysctls can be used for configuration:

```
net.inet.tcp.functions_available - List all available stacks
net.inet.tcp.functions_default - Set/Get default stack
```

What follows is a short example of a server-side setting '`mytcp`' as a stack to use on the listen socket. Each new connection it accepts would get '`mytcp`' as its TCP stack.

```
socklen_t slen;
char *stack_name="mytcp";
struct tcp_function_set fsn;
int error;

memset(&fsn, 0, sizeof(fsn));
strcpy(fsn.function_set_name, stack_name);
slen = sizeof(fsn);
error = setsockopt(s, IPPROTO_TCP, TCP_FUNCTION_BLK, &fsn, slen);
if (error)
printf("Could not set TCP stack to %s, error:%d\n", func_blk, errno);
```

TCPPCAP—A Debugging Feature

At times, TCP problems surface in weird ways, and we want to have access to the last few packets or TCP state transitions so that we can figure out what went wrong. This feature provides that by saving the last configurable number of packets in the corresponding TCP control block. The number of packets can be specified on a global, i.e., per-system basis, or per-connection using a socket option.

A note of caution: Although this feature does a good job of keeping a cap on the maximum amount of memory that can be allocated to it, the number should be chosen carefully, as saving packets consumes memory.

This feature can be enabled with the TCPPCAP option in the kernel configuration. Even then, the feature is deactivated. When activated, the TCP control block has `t_inpkts`, which is the list of input packets saved, and `t_outpkts`, which is the list of output packets saved.

The following sysctl can be used for configuration:
`net.inet.tcp.tcp_pcap_packets` - Enable feature and set number of packets to capture per connection in each direction
`net.inet.tcp.tcp_pcap_clusters_referenced_max` - Maximum `mbuf` clusters allowed to be referenced

The following sysctls can be used to monitor memory usage by this feature:
`net.inet.tcp.tcp_pcap_clusters_referenced_cur` - Total `mbuf` clusters referenced
`net.inet.tcp.tcp_pcap_alloc_reuse_ext` - Total reused `mbufs` with external storage
`net.inet.tcp.tcp_pcap_alloc_reuse_mbuf` - Total reused `mbufs` with internal storage
`net.inet.tcp.tcp_pcap_alloc_new_mbuf` - Total new `mbufs` allocated

Server-side TCP Fast Open (TFO)

TCP performs a three-way handshake (`3-WHS: SYN, SYN-ACK, ACK`) between client and server before any actual data can be transferred on the connection. For short-lived connections, this can be a significant part of the total time spent on the transaction, i.e., latency experienced by the client.

TCP Fast Open (TFO), introduced by rfc 7413, addresses this problem by allowing `SYN` and `SYN-ACK` packets to carry data if the client has a valid security cookie with which the server can authenticate the client.

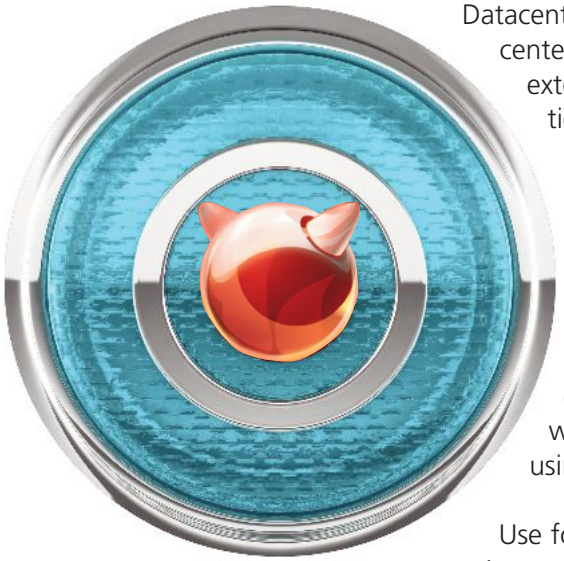
FreeBSD now has server-side support for this feature. The main part of TFO is the security cookie. The client first requests the cookie by sending `SYN` with a `FAST OPEN` option and an empty cookie field. The server generates a cookie and sends it back with the `FAST OPEN` option of a `SYN-ACK` packet. The client caches that cookie to use it in `FAST OPEN` connections to that same server. Once the client has the cookie, it sends `SYN` with the data and the cached cookie in the `FAST OPEN` option field. If the cookie is valid, the server acks both the cookie and the data and sends the data to the application. If the cookie is not valid, the server drops the data and only acks the `SYN`. If the server accepts the data provided in the `SYN`, it can send the response data before the `3-WHS` finishes. The client acks both the `SYN` and data if present. If not, it only acks the server's `SYN`. The rest of the connection proceeds as a regular TCP connection.

The client can make TFO connections until the cookie is valid and its validity is configurable on the server side. Thus, if a client is making a lot of short-lived connections to the same server, this feature can help a lot in reducing overall latency.

This feature can be enabled by adding '`options TCP_RFC7413`' to the kernel configuration. How many multiple concurrent keys to use can be specified with '`options TCP_RFC7413_MAX_KEYS=<num-keys>`' in the kernel configuration. Currently it defaults to `2` configuration sysctls:
`net.inet.tcp.fastopen.enabled` - Enable/Disable this feature
`net.inet.tcp.fastopen.autokey` - Automatic key renew timeout; defaults to 120secs

`net.inet.tcp.fastopen.acceptany` - Accept any key client supplied cookie; disabled by default
`net.inet.tcp.fastopen.keylen` - The key length in bytes
`net.inet.tcp.fastopen.maxkeys` - Max keys supported
`net.inet.tcp.fastopen.numkeys` - Total keys installed

Datacenter TCP



Datacenter TCP (DCTCP) is a congestion control mechanism to be used in datacenters. It uses explicit congestion notification (ECN) to estimate the extent/amount of congestion instead of just detecting that some congestion has occurred.

Traditional congestion control mechanisms underperform inside datacenters because the traffic is high throughput, low latency, and bursty in nature. Because ECN can estimate the severity of congestion, DCTCP uses that to reduce the congestion window accordingly, which helps in using available bandwidth efficiently.

DCTCP is now included as a congestion control module which can be the default congestion control for all connections or can be set per connection using socket options. The current implementation can also work in a one-sided fashion where only one side (sender or receiver) is using DCTCP.

Use following sysctls to configure this feature:
`net.inet.tcp.cc.dctcp.slowstart` - Determines whether to reduce the congestion window by half after the first slowstart.
`net.inet.tcp.cc.dctcp.shift_g` - Determines convergence time and bandwidth utilization. The IETF Draft suggests keeping it at '`4`' which we follow in the implementation.
`net.inet.tcp.cc.dctcp.alpha` - Determines how aggressive you want DCTCP to be at the cost of queueing delay at the beginning of the connection. The IETF Draft suggests being less aggressive with value '`1`,' but we decided to be more aggressive and kept the default at '`0`' in FreeBSD.

Packetization Layer Path MTU Discovery for Blackhole Detection

The maximum transmission unit (MTU) is the largest amount of data that can be sent in a frame. If a device is in the path with outgoing interface MTU less than the size of the frame, and if the '`don't fragment`' (DF) bit is set, the frame is discarded and an ICMP message '`Fragmentation needed and DF set`' is sent back to the sender with the outgoing interface MTU size in it. Upon receiving this message, the sender can adjust its MTU and resend the frame.

Sometimes these ICMP messages are blocked because of firewalls or some other mysterious reasons, and packets are silently discarded without the server ever knowing about them. That is called a PMTU blackhole. In such cases, being able to detect a blackhole without needing ICMP is really useful and the Packetization Layer Path MTU Discovery (PLPMTUD) blackhole detection introduced by rfc 4821 does exactly that.

When sending a packet fails twice with retransmission timeout (RTO), it tries to detect whether the connection is experiencing a blackhole along the path. It does this by reducing maximum segment size (MSS) to a preconfigured value. And we continue reducing the MSS in the case of more successive failed sending attempts until we reach the minimum configured MSS value. If the packet can be sent using a lower MSS, that means we, indeed, were experiencing a blackhole along the path. But if we cannot send the packet, even with the smallest configured MSS value, it is not the blackhole, but real congestion that is causing packet drops, and in that case, we restore the MSS to its original value.

Another important part of this feature is the upward probing of the network with incrementally

higher values of MSS on successful transfers. Yet, that is not currently implemented in FreeBSD, and because of that, this feature is disabled by default.

In many congestion control mechanisms, congestion window growth depends on the size of MSS, and therefore a connection in a blackhole may experience slower growth of a congestion window and thus be slower in utilizing path capacity.

Use the following sysctls to configure this feature:

`net.inet.tcp.pmtud_blackhole_detection` - Enable/Disable
`net.inet.tcp.pmtud_blackhole_mss` - Lowered MSS for IPv4
`net.inet.tcp.v6pmtud_blackhole_mss` - Lowered MSS for IPv6

Use the following sysctls to monitor this feature:

`net.inet.tcp.pmtud_blackhole_activated` - How many times we detected a blackhole and activated MSS down-probing.
`net.inet.tcp.pmtud_blackhole_activated_min_mss` - How many times we detected a blackhole and went down to minimum MSS configured during MSS down-probing.
`net.inet.tcp.pmtud_blackhole_failed` - How many times we detected a blackhole incorrectly

Short-lived TCP Connections Scalability Effort

The layered design of the networking stack and its implementation indicate complex locking and serialization challenges. Each layer has to talk to the neighboring layers to stay in sync to accept, serve, and tear down connections. Even within each layer, depending on its functionality, there has to be proper synchronization to maintain consistency when many, many connections are sharing the same data structures and memory resources.

Struct inpcb is the IP layer protocol control block structure which captures the network layer states like host addresses and routing information for TCP, UDP, and the like.

Before this effort/change for short-lived connections, close to 80% of received packets were processed holding exclusive write lock. This means that only one cpu core can work on receiving packets and the rest of the cores have to wait for the lock to be released. Now, a new `INP_LIST` lock has been added that protects modifications to the global inpcb list. This allows `INP_INFO_RLOCK` (a read lock) in critical paths like input processing, and only occasionally needs `INP_INFO_WLOCK` (a write lock) when it is really needed, i.e., walking global inpcb list while being stable.

The maximum number of TCP connections (setup and teardown) per connection is increased from 60k/sec to 150k/sec.

Improved Protection Against Blind In-window SYN/RST Spoofed Attack

If an attacker could somehow guess the valid window of sequence numbers and forge a `SYN` or a `RST` packet, FreeBSD used to drop the connection without looking at whether it's the exact packet it expects at that point in time or not.

Now with the enhancements proposed by rfc 5961, if the incoming `RST` is the exact sequence number we are expecting, drop the connections, and if it's not that, but within the window, send a challenge `ACK`. For incoming `SYN`, if the connection is in a synchronized state, send a challenge `ACK`.

Sending a challenge `ACK` makes a genuine sender generate a `RST` that matches the exact sequence number that we expect. Basically, with this mechanism in place, it's very hard for the attacker to spoof a valid `SYN/RST` that can cause us to drop the connection. Some middleboxes or broken TCP stacks may still respond to the challenge `ACK` with a `RST` packet that has a SEQ number that we do not expect. In that case, because of this feature, we may see a lot of spurious challenge `ACK` and `RST`s going back and forth. Challenge `ACK`s should be throttled to alleviate this problem so that we only send a limited amount of challenge `ACK`s per time period. For example, send only 10 challenge `ACK`s in a 5-second period. The time and amount of acks allowed should both be tunables. This is not yet implemented in FreeBSD.

This feature is enabled by default on FreeBSD. It can be disabled by setting `net.inet.tcp.insecure_rst` and `net.inet.tcp.insecure_syn` to '1'.

TCPDEBUG Like Information with Dtrace Trace Points

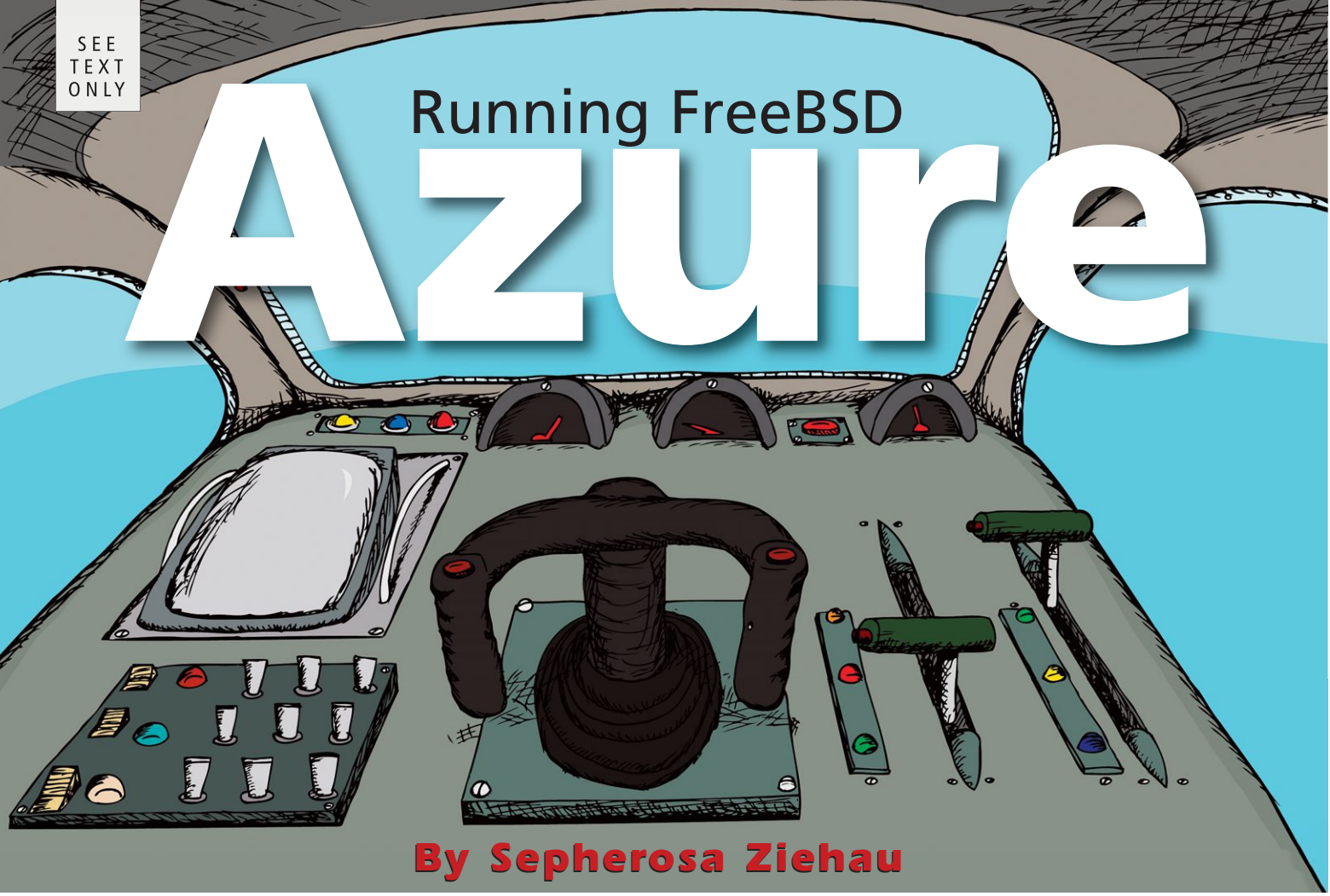
TCP has a debugging functionality that can print traces of connection state changes and other useful events like sending and receiving packets. But to enable this feature, the kernel needs to be built with the TCPDEBUG option in the kernel configuration file and the associated socket needs to be marked with the `SO_DEBUG` option—which means recompiling the application.

Now, with Dtrace (a dynamic tracing framework) probe points enabling the same functionality, it can be used without needing to recompile the kernel or userland application. `$src/share/dtrace/tcpdebug` is a sample script and a nice example of how to use this feature.

HIREN PANCHASARA has been struggling his way through kernel code for a few years now, and lately more so in networking and TCP. He runs FreeBSD on all things possible and wishes to bring more and more newbies to FreeBSD by mentoring and other means.

SEE TEXT ONLY

Running FreeBSD Azure



By Sepherosa Ziehau

On June 8, 2016, a standard FreeBSD 10.3 image was published into the Azure Marketplace. Microsoft published the image working as part of the FreeBSD community and in collaboration with the FreeBSD Foundation. This was a milestone representing the culmination of several years of Microsoft collaboration with the FreeBSD community. FreeBSD is leveraged as the base OS for a number of virtual appliances running in Azure, and so Microsoft has a natural interest in making sure it runs well.

How It Began

Microsoft's interest in FreeBSD started with feedback from customers and partners. Customers want to run a wide variety of operating systems, including Windows, Linux, and FreeBSD, both in on-premises environments and in the Azure public cloud. Partners have quite a lot of FreeBSD-based virtual appliances, so, of course, Microsoft wants its partners' products to run well on Hyper-V and in Azure in order to give Microsoft's customers a full range of choices. To provide this variety of operating systems, Microsoft contributes to the relevant open-source communities. Specifically, for FreeBSD, Microsoft has developers working on the code to make sure that FreeBSD runs great on Hyper-V and in Azure.

When It Began

The initial work on FreeBSD for Hyper-V/Azure was the result of collaboration between Microsoft, NetApp, and Citrix Systems. By the end of 2011, Microsoft teamed up with NetApp and Citrix Systems to bring the "enlightened" Hyper-V device drivers to FreeBSD. Though there were challenges due to the differences between Microsoft's coding style and FreeBSD's style(9), and challenges in hooking up the device drivers to FreeBSD's device(9), this collaboration was quite successful: all of the major "enlightened" Hyper-V device drivers, i.e., bus driver, storage driver, and network driver, started to work in FreeBSD on Hyper-V/Azure. It also led to a joint presentation of FreeBSD on Hyper-V/Azure at

BSDCan 2012: http://www.bsdcan.org/2012/schedule/events/287.en.html. After the initial work on these "enlightened" device drivers, Microsoft developers made additional progress, and in 2013, the drivers were imported into the main FreeBSD source code tree.

Keep the Ball Rolling

Microsoft saw increasing market demand for FreeBSD on Hyper-V/Azure, and requirements to make the FreeBSD on Hyper-V/Azure performant and feature-rich to the same degree as Windows and Linux. So in the middle of 2014, Microsoft formed a team in Shanghai to focus on FreeBSD on Hyper-V/Azure. Since most of the developers came with a Linux background, they took some time to familiarize themselves with the FreeBSD on Hyper-V/Azure code before getting their hands dirty with further development. And with help from FreeBSD's Xin Li (delphij@), various drivers for Hyper-V/Azure utilities—e.g., shutdown and KVP (key-value-pair)—were imported into the FreeBSD source code tree.

The First Monument for FreeBSD on Hyper-V/Azure

At the Microsoft TechEd 2014 conference (it is now called Microsoft Ignite), for the first time, FreeBSD on Hyper-V was formally mentioned by Microsoft in the Virtualizing Linux and FreeBSD Workloads on the Next Release of Windows Server Hyper-V talk. At that time, several FreeBSD images had already been imported into Azure's VM Depot by FreeBSD's Glen Barber (gjb@).

The Whirling Wheel

At the end of 2014, Microsoft teamed up with NetApp again to improve the "enlightened" storage device performance on FreeBSD. The result of this collaboration was quite exciting, producing impressive gains in storage device performance. This code was contributed to the FreeBSD source-code tree without any reservations in early 2015.

The Steep Learning Curve, and We Came, We Saw, We Conquered

After concluding the previous stage of storage device improvement, Microsoft decided to improve the performance of the "enlightened" network device driver. The development of checksum offloading and TCP segmentation offloading progressed smoothly. However, as mentioned before,

most developers on the team came with a Linux background, and they had a hard time with the steep learning curve for the interaction between the FreeBSD network stack and the "enlightened" network device driver. That uphill battle lasted for more than half a year. More and more developers joined the combat. After the "enlightened" network device driver went through a series of serious surgeries, FreeBSD VMs could finally drive full line rate on a 10-Gbps physical network, and drive pretty decent performance on 40-Gbps physical network. The development team recorded this journey of "exit out of the rabbit hole" in a presentation at BSDCan 2016: http://www.bsdcan.org/2016/schedule/events/681.en.html.

The Second Monument for FreeBSD on Hyper-V/Azure

At BSDCan 2016, Microsoft announced the availability of FreeBSD 10.3 globally in the Azure Marketplace, along with many FreeBSD-based Azure virtual appliances, e.g., Citrix Systems' Netscaler and Netgate's pfsense.

We Are Moving On

More development for FreeBSD on Hyper-V/Azure is in the works, including, but not limited to, further improving the performance of "enlightened" storage device driver SR-IOV and many others. Microsoft also continues to help FreeBSD-based Azure virtual appliance vendors and Hyper-V users make their products run performant and stably. ●

SEPHEROSA ZIEHAU has been a FreeBSD src committer since 2007. He helped Microsoft on their Hyper-V FreeBSD development early this year, and later joined Microsoft to keep the Hyper-V FreeBSD development moving.

THE INTERNET NEEDS YOU

GET CERTIFIED AND GET IN THERE!

Go to the next level with BSD CERTIFICATION

Getting the most out of BSD operating systems requires a serious level of knowledge and expertise

NEED AN EDGE?

BSD Certification can make all the difference.

Today's Internet is complex. Companies need individuals with proven skills to work on some of the most advanced systems on the Net. With BSD Certification

YOU'LL HAVE WHAT IT TAKES!

SHOW YOUR STUFF!

Your commitment and dedication to achieving the **BSD ASSOCIATE CERTIFICATION** can bring you to the attention of companies that need your skills.

BSDCERTIFICATION.ORG

Providing psychometrically valid, globally affordable exams in BSD Systems Administration

SEE TEXT ONLY

FreeBSD Toolchain

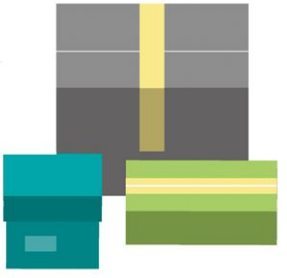
As with other BSD distributions, FreeBSD has the concept of a base system, an integrated kernel, and core userland, which are developed, tested, and released together by a common team. The userland includes the C language runtime, the build toolchain, basic system utilities, and some third-party libraries and applications. The toolchain includes the compiler and linker, which translate source code into executable objects, and related utilities for inspecting or modifying those objects.

In addition to this base system, there's a third-party software ecosystem managed through the FreeBSD ports tree and binary packages. Over 26,000 third-party software packages exist in the ports tree.

For most of its history FreeBSD relied on the full GNU toolchain suite, including the GNU Compiler Collection (GCC) compiler and GNU binutils. They served the FreeBSD Project well from 1993 until 2007, when the GNU project migrated to releasing its software under the General Public License, version 3 (GPLv3). In contrast to version 2 of the license, GPLv3 places a number of new restrictions on software users, and FreeBSD developers and users found these objectionable. As a result, the GNU toolchain in FreeBSD was not updated to a new upstream version and quickly became outdated.

Beginning shortly after that transition, a number of developers in the FreeBSD Project started updating components of the FreeBSD toolchain to modern, copy-free alternatives. Some aspects of the toolchain are architecture-dependent, and in general the tools described below apply to FreeBSD's Tier-1 architectures.

BY
ED
MASTE



Compiler

The LLVM project is a collection of modular and reusable components used to build compilers and other toolchain utilities. It began as a research project at the University of Illinois and has since grown to become a production-quality tool set used in proprietary and open-source software and academic settings. Clang is the C and C++ compiler included in the LLVM suite.

We've experimented with Clang in FreeBSD since 2009, starting with a snapshot from the upstream project's Subversion repository. When first added, it was made available as an option to facilitate testing and development, but it was not enabled as the default compiler for some time. When FreeBSD 10 released in January 2014, it included Clang as the system compiler (that is, installed as `/usr/bin/cc`) for the 32- and 64-bit Intel x86 architectures. It was later made the default for the 32-bit FreeBSD ARM port. The 64-bit ARM port began with Clang as the system compiler.

Clang includes some support for the MIPS and PowerPC architectures, but is not yet capable of replacing the system compiler there; those architectures continue to use GCC.

FreeBSD 11 includes Clang 3.8.0, and work is underway in FreeBSD-CURRENT to update to Clang 3.9.0.

Linker

The linker takes as input individual object files produced by a compiler or assembler and links them into an executable binary or library. To date we've used the GNU linker in FreeBSD, most recently updated to version 2.17.50 in February 2011. This linker lacks link-time optimizations and support for certain debugging features.

GNU ld in the base system also lacks support for AArch64 (arm64) and RISC-V, two CPU architectures recently added to FreeBSD. Building FreeBSD for these currently requires the linker be installed from the ports tree or as a package, in which case it will be used automatically.

The LLVM family of projects includes a linker: LLD. It is a high-performance linker with support for ELF, COFF, and Mach-O. Where possible, it maintains command-line and functional compatibility with existing linkers such as GNU ld, but LLD's authors are not constrained by strict compatibility where it hampers performance or desired functionality.

Work began on LLD's ELF support in July 2015 and has progressed very quickly; it currently self-hosts on several of LLVM's supported architectures and includes almost all of the functionality required to build the FreeBSD base system.

In comparison with GNU ld, in FreeBSD LLD will provide AArch64 and eventually RISC-V support, full-program Link-Time Optimization (LTO), support for new Application Binary Interfaces (ABIs), other linker optimizations, debugging features, and much faster link times.

As with early experiments with Clang, LLD will first be added to FreeBSD as an option and will not be installed as the system linker (`/usr/bin/ld`). It will be available by adding `-fuse-ld=lld` to the compiler's command line arguments (for example, via the `CFLAGS` variable).

We've included LLD 3.9 alongside the Clang 3.9 update and expect it to be available in FreeBSD-CURRENT in October or November 2016. It is expected to be added to FreeBSD 11.1 as well.

Binary Utilities

The toolchain includes a number of small binary utilities for inspecting or modifying object files, binaries, and libraries. These are tools that report information about an object, binary, or library, or transform an object's contents or format. These tools were historically provided by GNU binutils.

The ELF Tool Chain Project maintains BSD-licensed implementations of essential compilation tools and libraries for working with ELF object files and binaries, and DWARF debugging information. It began as part of the FreeBSD Project, but became a standalone project in order to facilitate collaboration with the wider open-source community.

For FreeBSD 11 we have migrated to ELF Tool Chain's implementation of `addr2line`, `c++filt`, `objcopy`, `nm`, `readelf`, `size`, `strip`, and `strings`. The `libelf` and `libdwarf` libraries also come from ELF Tool Chain.

ELF Tool Chain also includes a version of FreeBSD's `ar`, `brandelf`, and `elfdump` utilities. We will migrate to those in the future if they gain compelling new features or improvements.

Runtime Libraries

A number of libraries are required to provide runtime support for the toolchain. The compiler-rt library comes from the LLVM project and includes several components. Low-level target-specific hooks required by GCC or Clang's code generation are known as "builtins." These are routines that perform operations that the compiler will not emit directly into the output object code.

Compiler-rt includes sanitizer runtimes that provide Clang's runtime support for identifying

erroneous software behaviour. AddressSanitizer (asan) detects out-of-bounds access to heap, stack, and globals, use-after-free and related errors, and double and invalid free. It is available by adding `-fsanitize=address` to the compiler's command line.

The UndefinedBehaviourSanitizer (ubsan) catches undefined behaviour; that is, an operation for which the language does not have specified behaviour. Examples include overflowing integer operations (such as addition and bit shifting), division by zero, and using uninitialized variables. The `-fsanitize=undefined` command line argument enables ubsan.

Clang includes other sanitizers in the upstream repository, including ThreadSanitizer for detecting data races in multithreaded applications and MemorySanitizer for detecting uninitialized reads. Work is underway to make these available with the system compiler in a later FreeBSD version.

For C++ runtime support we use a combination of PathScale's libcxxrt for low-level Application Binary Interface (ABI) support. This includes Run-Time Type Information (RTTI), exception handling, and thread-safe initializers. The C++ standard library is LLVM's libc++.

Debugger

A replacement debugger also comes from the LLVM project: LLDB. As with the rest of LLVM, LLDB is built as a set of reusable components, and builds on other parts of LLVM and Clang. LLDB has access to a full Clang compiler, which it uses as the expression parser. This provides high fidelity in interpreting the user's expressions: LLDB is capable of parsing any expression valid in the original source code being debugged.

LLDB is included in FreeBSD 11 for the 32- and 64-bit ARM architectures, and 64-bit Intel (amd64).

In addition to LLDB in the base system, the GNU GDB debugger is available by building from the ports tree or installing the package. It has been updated and includes improved threading support and kernel debugging. GDB 7.11.1 is available with a simple `pkg install gdb` command.

DTrace

FreeBSD 11 includes support for using Compressed Type Format (CTF) data in userland. This means that userland Function Boundary Tracing (FBT) probes can have typed arguments when the corresponding executable or libraries are compiled with CTF. The FreeBSD base system build infrastructure includes support for compiling with CTF with a setting in the `/etc/src.conf` file.

FreeBSD 11 also adds support for Statically Defined Tracing (STD) probes in userland. A `.d` file containing probe definitions can be included in the list of source files, and the build infrastructure will automatically generate a header containing probe macros that can be referenced in source files.

Future Work

Work is underway on a number of additional toolchain components which will become available in FreeBSD in the future. LLVM provides a code coverage tool llvm-cov, which can operate in a mode similar to GNU gcov, and can operate with Clang's instrumentation-based profiling. LLVM also provides an OpenMP runtime (libomp), an Application Programming Interface (API) for shared memory multiprocessing.

With LLD as the linker, two GNU binutils tools remain: the assembler (as) and objdump. There is currently no candidate to replace as. However, it is actually not required to build the FreeBSD base system on Tier-1 architectures: Clang's integrated assembler is used to build assembly source files. GNU as could simply be removed, or a GNU as-compatible driver could be built from the integrated assembler. Objdump is also not required by the base system build and could be removed. LLVM provides an llvm-objdump that is currently limited in functionality, but will likely be a viable replacement in due course.

Ongoing efforts in the FreeBSD community and in the upstream Clang/LLVM project are attempting to bring the Clang, LLD, and LLDB suite to the lower-tier FreeBSD architectures that are still using GCC today. ●

ED MASTE manages project development for the FreeBSD Foundation and works in an engineering support role with the University of Cambridge Computer Laboratory. He is also a member of the elected FreeBSD Core Team. Aside from FreeBSD and ELF Tool Chain, he is a contributor to a number of other open-source projects, including LLVM, QEMU, and Open vSwitch. He lives in Kitchener, Canada, with his wife, Anna, and sons, Pieter and Daniel.

THROUGH DECEMBER 2016 BY DRU LAVIGNE

2016 Events Calendar

The following BSD-related conferences will take place in November and December 2016. More information about these events, as well as local user group meetings, can be found at **www.bsdevents.org**.

Meet BSD California • Nov 10–12 • Berkeley, CA

https://meetbsd.com • The 5th biennial MeetBSD conference will be held at UC Berkeley. This conference provides a mix of formal presentations, unconference activities, and opportunities to network with other BSD users and developers. The BSDA certification exam will be available at this event. Registration is required to attend this conference.

LISA 16 • Dec 4–9 • Boston, MA

https://www.usenix.org/conference/lisa16/ • The 30th Large Installation System Administration conference will be held at the Sheraton Boston. There will be a FreeBSD booth in the expo area. Registration is required to attend the conference and a nominal fee is required to attend the expo.

SEE TEXT ONLY

new faces of FreeBSD

BY DRU LAVIGNE

This column aims to shine a spotlight on contributors who recently received their commit bit and to introduce them to the FreeBSD community. This month, the spotlight is on Eric Badger and Toomas Soome, who became src committers in July and August 2016.

Tell us a bit about yourself, your background, and your interests.


ERIC

Eric: I'm a software engineer at Dell Compellent, based in Minnesota, USA. I work on the Platform OS team in charge of the FreeBSD operating system running on Dell Storage controllers. I've been doing this for a little more than two years. I've learned a ton about FreeBSD working at Dell, partly from solving problems that arise in the OS and partly from the various smart people that I get to work with.

I'm interested in lots of things; nearly anything low level-ish can capture my interest. The things I've actually had time to work on have been mostly oriented towards networking, debugging tools (like gdb or DTrace), and storage devices. I just bought a BeagleBone Black for playing around with FreeBSD on something other than amd64, and have had lots of fun doing that.

Outside of computers, I like to play music (I like most kinds and have played with bands here and there) and study languages (Spanish is my principal interest at the moment).

Toomas: I was introduced to Unix as a student at the University of Tartu in Estonia, back in about 1992. We had a mix of different systems back then: SCO, SunOS, Irix, later Ultrix, NetBSD, FreeBSD, and eventually also Linux, just to name few. At that time I was working for the university as a system administrator and that time meant a lot of compiling, software integration, and so on as all the (free) software was distributed as source.


TOOMAS

For a fun fact, we actually started with www using the CERN sources for both the server and browser.

Later on I was working as pre-sales engineer for a local Sun Microsystems reseller, which meant less compiling but still a lot of integration, architecting, and deep diving into almost every layer of IT infrastructure.

For other interests, snowboarding in winter and skydiving in summer. Although, due to time issues, those activities have been somewhat diminished.

How did you first learn about FreeBSD and what about FreeBSD interested you?

Eric: I encountered FreeBSD at my first computer job at the Supercomputing Institute while a student at the University of Minnesota. My first boss was the network admin, and she used FreeBSD for network management tools and jump boxes onto the switch management network. I wanted to learn as much as possible about how things worked, so I added a FreeBSD box to my home network of Linux boxes (mainly old computers from friends and family). That first FreeBSD machine was my DHCP server and also ran tftpd and NFS for serving up files to PXE-booted machines.

The division between base and ports definitely stood out to me. I ran mostly Debian and CentOS then, which meant that sometimes third-party software wasn't available in the repos, or wasn't available at the version I wanted. FreeBSD ports meant I could have relatively bleeding edge software on a relatively stable base. Since working with FreeBSD as a developer, I've come to appreciate the fact that kernel and userland all live in the same tree. This often makes understanding and working with

things that cross the boundary easier. I've also come to appreciate the first class treatment of source code. FreeBSD makes it easy to get the system source and rebuild parts of it on your system. Ports make it easy to hack on third party software that needs a little help to build on FreeBSD.

Toomas: Cannot remember the year–it was a long time ago when I was still with the university, but we had a few instances of FreeBSD, and NetBSD. But BSD-based systems have always been interesting due to their roots and now even more as one of the Unix brand providing free software and really supporting technology exchange.

How did you end up becoming a src committer and which part of src do you work on?

Eric: I submitted a couple of patches that had been accepted to the tree and started a few discussions, mainly on -current. I think I earned my stars mainly by debugging and characterizing tricky problems, however, rather than a large number of patch submissions.

Mostly I've touched code under sys/kern at this point. I wouldn't say I've developed any strong specialty yet, though I have a growing list of things that I'd like to get into a committable state, including improved support for FreeBSD on KVM, some SR-IOV work, and a few other things.

Toomas: The roots are in a different operating system, due to my own roots to technologies related to Sun Microsystems. I was poking around with Illumos-based systems and eventually found myself investigating ways to improve boot loader and related topics. While working with different options, I found myself investigating the FreeBSD loader and ended up porting it to Illumos. That work has reached the stage where we are very close to implementing an actual integration. Quite obviously, Illumos has its own needs for its boot loader and so gradually those needs got addressed and some changes and improvements were implemented in the port. Once you have done some changes, you soon find that it is reasonable to offer some of those back upstream. Eventually, I guess, people with commit rights got the idea to invite me to the ranks of FreeBSD developers, which was quite a nice surprise for me.

So the area I'm working on is the boot loader and related topics. Some results are already integrated into FreeBSD, hopefully for more good than damage, but there are still many issues to investigate regarding how is the best way to improve the boot loader.

How has your experience been since joining the FreeBSD Project? Do you have any advice for readers who may be interested in also becoming a src committer?

Eric: My experience has been good. It's kind of like having a second job, and in that respect it can be demanding. I've only a couple of commits at this point, but still spend a lot of time playing with things and trying to understand them. But it feels like time well spent, since I'm usually having fun and have the opportunity to contribute to a major open-source operating system.

My advice to a potential future committer: if you find something that doesn't seem to be working correctly in FreeBSD, don't stop until you understand why. I find that there are few better ways of getting to understand something than debugging a problem you've personally experienced. Working with a company that uses FreeBSD also goes a long way. At Dell Compellent, there are hundreds of FreeBSD machines being pounded on by our test group every day, and any OS issues come through my group. That has provided a tremendous learning opportunity.

Toomas: I have only a positive experience. The most valuable part is really about information exchange, to have more ways to learn how the different components are related, how teams are related, and what are the driving forces behind a decision. Most of the success of the particular project is really not about how well some piece of code is written, as there is always space for improvements, but how well it will fit in with the rest of the system and if you can somehow manage to bring in some feature at the right time to provide the most value for the system or other teams or people.

As for advice, you may have all the dreams and training, but before the door opens and you actually make your step through, nothing will matter. Once you have done the first step, and can see the training you have received is actually working and giving results, then you are probably sold and the hard work towards the goals you have set will really start. So, make that first step and check out if you like what you are experiencing. ●

Dru Lavigne is a Director of the FreeBSD Foundation and Chair of the BSD Certification Group.

SEE TEXT ONLY

svn UPDATE

by Steven Kreuzer

On July 8, the head of svn was merged into a stable/11 branch, and when the source repository was thawed the following day, FreeBSD-CURRENT could affectionately be referred to as FreeBSD 12. While both developers and end users are busy testing and helping squash last-minute bugs to make sure that 11-RELEASE is as rock solid as any other release, FreeBSD 10 is still actively being developed and quite a few interesting commits focusing on improved performance and compatibility have made their way from head to stable/10.

MFH r301732, r302288 (https://svnweb.freebsd.org/base?view=revision&revision=302931)

`Console="comconsole"` has been switched to `boot_multicons="YES"` in EC2. This allows FreeBSD instances running in Elastic Compute Cloud to take advantage of an API (https://aws.amazon.com/blogs/aws/ec2-instance-console-screenshot/) that was recently introduced by Amazon to allow for the capturing of screenshots of an emulated VGA device.

Indirect segment I/Os have been enabled by default in the Xen blkfront driver when running on EC2. Due to improvements in EC2, the performance penalty that was present on some EC2 instances no longer exists, and enabling this feature now consistently yields ~20% higher throughput with equal or lower latency.

MFC r297187 (https://svnweb.freebsd.org/base?view=revision&revision=302901)

Intel PRO/1000 gigabit Ethernet adaptor now supports checksum offloading for TCP/IPV6 and UDP/IPV6 as well as SCTP checksum offloading for SCTP/IPV6.

MFH r297023 (https://svnweb.freebsd.org/base?view=revision&revision=302800)

Kldstat(8) now has the ability to print out the module-specific information in likely formats. Among other things, this gives us the ability to find out the syscall number of a dynamically loaded syscall that has a dynamically allocated vector number.

MFC r302402 (https://svnweb.freebsd.org/base?view=revision&revision=302791)

The ahci(4) driver can now attach to a controller with 32 ports. Due to an incorrect sign, expansion in variables that are supposed to be bit fields resulted in an infinite loop. Fixing this allows a system to properly detect 32 devices configured on an AHCI HBA in bhyve.

MFC r302265, r302382 (https://svnweb.freebsd.org/base?view=revision&revision=302714)

The ZFS ARC min and max values can now be tuned at runtime. Prior to this change, the ARC min and max values could only be changed using boot time tunables. It is now possible to change these values at runtime using the sysctls and `vfs.zfs.arc_max and vfs.zfs.arc_min`

MFC r302174 (https://svnweb.freebsd.org/base?view=revision&revision=302688)

The output of "`sysctl vm.vmtotal`" on machines configured with more than 2TB of virtual memory has been fixed.

MFC r301545 (https://svnweb.freebsd.org/base?view=revision&revision=302270)

SeR-IOV guest support has been added to the mlx5en driver. This adds the missing pieces needed for device setup using the mlx5en driver inside a virtual machine that is providing hardware access through SR-IOV.

STEVEN KREUZER is a FreeBSD Developer and Unix Systems Administrator with an interest in retro-computing and air-cooled Volkswagens. He lives in Queens, New York, with his wife, daughter, and dog.

SEE TEXT ONLY

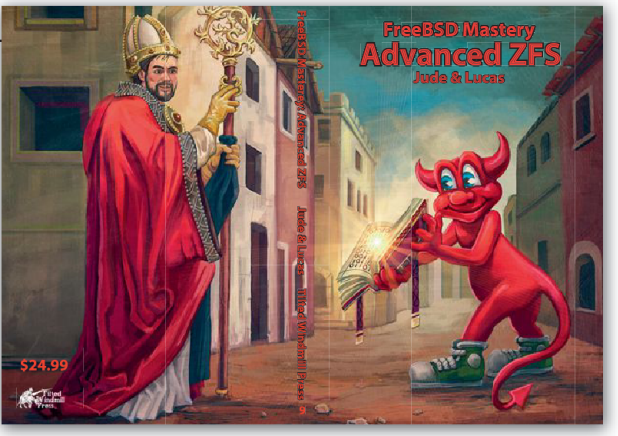
BOOKreview

by Joseph Kong

FreeBSD Mastery: Advanced ZFS

by Allan Jude and Michael W. Lucas

Publisher ..... *Tilted Windmill Press*
ISBN-10: *0692688684*
ISBN-13: *978-0692688687*
Pages *242*



FreeBSD Mastery: Advanced ZFS by Allan Jude and Michael W. Lucas is a clear and concise tour of the more complicated and esoteric parts of managing the Z File System (ZFS). This book is directly on target and does not waste your time. The authors assume you're already familiar with ZFS pools, datasets, snapshots, clones, and so on, and that you're here specifically for the advanced stuff.

Chapter 1, Boot Environments, describes using ZFS to create Solaris-style boot environments, which are bootable backups of your operating system's kernel and userland, enabling you to easily revert changes.

Chapter 2, Delegation and Jails, describes ZFS's delegation system, which allows you to specify the commands a user or group can issue on each dataset. A discussion on how this delegation system works with FreeBSD jails is also included.

Chapter 3, Sharing Datasets, discusses ZFS's network file-sharing features. It walks you through FreeBSD's iSCSI and NFS implementations and how they relate to ZFS.

Chapter 4, Replication, describes how to create an exact copy of your filesystem elsewhere—for example, on another dataset in your pool, a second pool on your system, an external drive, a remote system, a tape, or just a file.

Chapter 5, ZFS Volumes, goes over the basics of ZFS volumes (zvols) and then delves into some common pitfalls.

Chapter 6, Advanced Hardware, covers different types of hardware and how they interact with ZFS, including SCSI enclosures, Host Bus Adapters (HBAs), SAS Multipath, Solid State Disks (SSDs), and Non-Volatile Memory Express (NVMe).

Chapter 7, Caches, describes ZFS's caching mechanisms, including an in-depth discussion of ZFS's Adaptive Replacement Cache (ARC), which is actually the bulk of the chapter.

Chapter 8, Performance, details how to improve ZFS's performance for your specific environment. This is quite an in-depth chapter and is my favorite of the book. It discusses how to assess ZFS's performance, ZFS's prefetch system, transaction group (txg) tuning, I/O scheduling, I/O queues, throttling writes, and more.

Chapter 9, Tuning, is a continuation (of sorts) of chapter 8 and describes how to adjust ZFS for your environment. I particularly enjoyed the discussion on how ZFS interacts with MySQL and PostgreSQL.

Chapter 10, ZFS Potpourri, contains an assortment of short guides on using ZFS, including splitting mirrored pools into multiple identical pools, deleting multiple neighboring snapshots, recovering destroyed pools, using the ZFS debugger (zdb), examining datasets in detail, and more.

Like Lucas's other titles, humor is ever-present in this book. For example, on page 195, there are four footnotes that contain a back-and-forth argument between the two authors. I found these tidbits helped break up the dense material; however, some readers may dislike them.

Bottom line, *FreeBSD Mastery: Advanced ZFS* is a superb book for the right reader. If you've already mastered the basics of ZFS and want to learn more, this book is for you. ●

JOSEPH KONG is a self-taught computer enthusiast who dabbles in the fields of exploit development, reverse code engineering, rootkit development, and systems programming (FreeBSD, Linux, and Windows). He is the author of the critically acclaimed *Designing BSD Rootkits* and *FreeBSD Device Drivers*. He is currently a senior software engineer for EMC's Isilon division. For more information about Joseph Kong visit www.thestackframe.org or follow him on Twitter @JosephJKong.

Support FreeBSD®

Donate to the Foundation!

You already know that FreeBSD is an internationally recognized leader in providing a high-performance, secure, and stable operating system. It's because of you. Your donations have a direct impact on the Project.

Please consider making a gift to support FreeBSD for the coming year. It's only with your help that we can continue and increase our support to make FreeBSD the high-performance, secure, and reliable OS you know and love!

Your investment will help:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure

Making a donation is quick and easy.
freebsdfoundation.org/donate

FreeBSD™
FOUNDATION